Jérémie Rafinesque

jeremie.rafinesque@imt-atlantique.net

Guillaume Chiquet

guillaume.chiquet@imt-atlantique.net

Arnaud Rohé

arnaud.rohé@imt-atlantique.net

Pierre Dugast

pierre.dugast@imt-atlantique.net

SIT213 Fil Rouge

Rapport Étape 5

Version 1,0 - 07/10/2019

Formation d'ingénieur

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

SOMMAIRE

Index des illustrations

I. INTRODUCTION

Tous les quatre étudiants au sein de l'IMT Atlantique, nous allons vous présenter notre deuxième étape du projet SIT213 au sein de ce livrable. Ce projet va nous permettre d'associer apprentissages du langage Java et simulation de signaux et briques de transmission.

L'objectif final de ce projet consiste à transmettre un message d'un point d'entrée à un point de sortie via un canal de transmission. Nous utiliserons lors de ce projet les connaissances acquises dans les modules SIT 211 (modélisation et validation des logiciels) et SIT 212 (simulation des signaux et briques de transmission).

Ce livrable présente l'ensemble des évolutions à chaque séance de notre projet.

II. LA RÉALISATION GLOBALE ATTENDUE

III. ÉTAPE 1

III.1 Objectif

- ✔ Réaliser une transmission élémentaire « back to back » ;
- ✔ Développer des composants dans le langage de programmation Java ;
- ✔ Mettre en œuvre ces composants à travers une classe de test sous la forme d'une chaîne de transmission.

III.2 Cahier des charges

Pour cette première itération du projet, le but est de réaliser une transmission élémentaire « back-to-back ». Elle est caractérisée par un transmetteur logique « parfait » ; on simule donc un canal de transmission parfait qui n'introduit pas de bruit parasite. On s'attend donc à recevoir un message identique à celui émis. Par conséquent le taux d'erreurs binaire du système doit valoir 0.

Voici l'illustration de la chaîne de transmission à l'étape 1 :

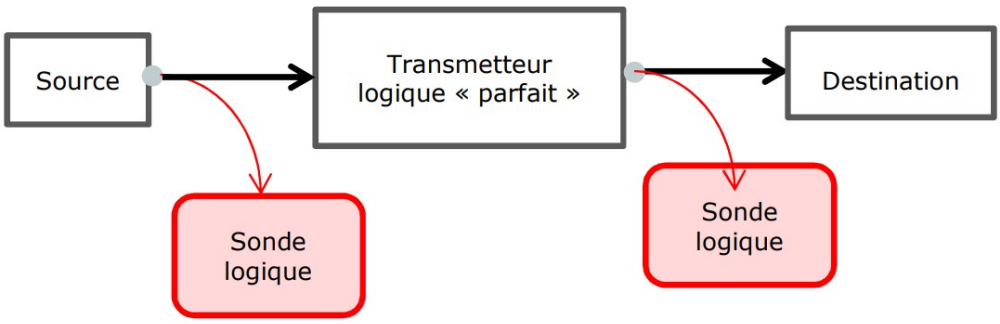

Illustration 1: Simulation Destination Finale

III.3 Développement Logiciel

III.3.1 Réalisation

- Développement de la classe « SourceAléatoire » afin de générer une chaîne Booleenne d'une longueur donnée de bits aléatoires, dans un objet Information ;
- Développement de la classe « SourceFixe » afin de générer une chaîne Booleenne à partir d'une chaîne de caractère donnée composée de '0' et de '1', dans un objet Information ;
- Développement de la classe « TransmetteurParfait » permettant de transmettre simplement un objet Information ;
- Développement de la classe « DestinationFinale » permettant de recevoir simplement un objet Information.

Modification sur l'existant :

- Implémentation de la méthode calculTauxErreurBinaire() de la classe Simulateur permettant de calculer le taux d'erreur binaire en comparant les objets Information ;
- Implémentation de la méthode execute() de la classe Simulateur, afin de rajouter l'option d'affichage des sondes ;
- Implémentation du constructeur Simulateur() de la classe Simulateur, afin de gérer les différentes méthodes de création de donnée binaire, et d'assembler les composant suivant le cahier des charges ci-dessus.

III.3.2 Résultats

Comme escompté, nous recevons bien un message identique à celui émis. Par conséquent le taux d'erreurs binaire du système vaut bien 0. Ces résultats sont tout à fait logiques et conformes à ce qui était attendu.

- Pour illustrer voici un exemple d'exécution avec l'option d'exécution **-mess 1234 -s ;** c'est-à-dire la génération d'un message aléatoire de longueur 1234

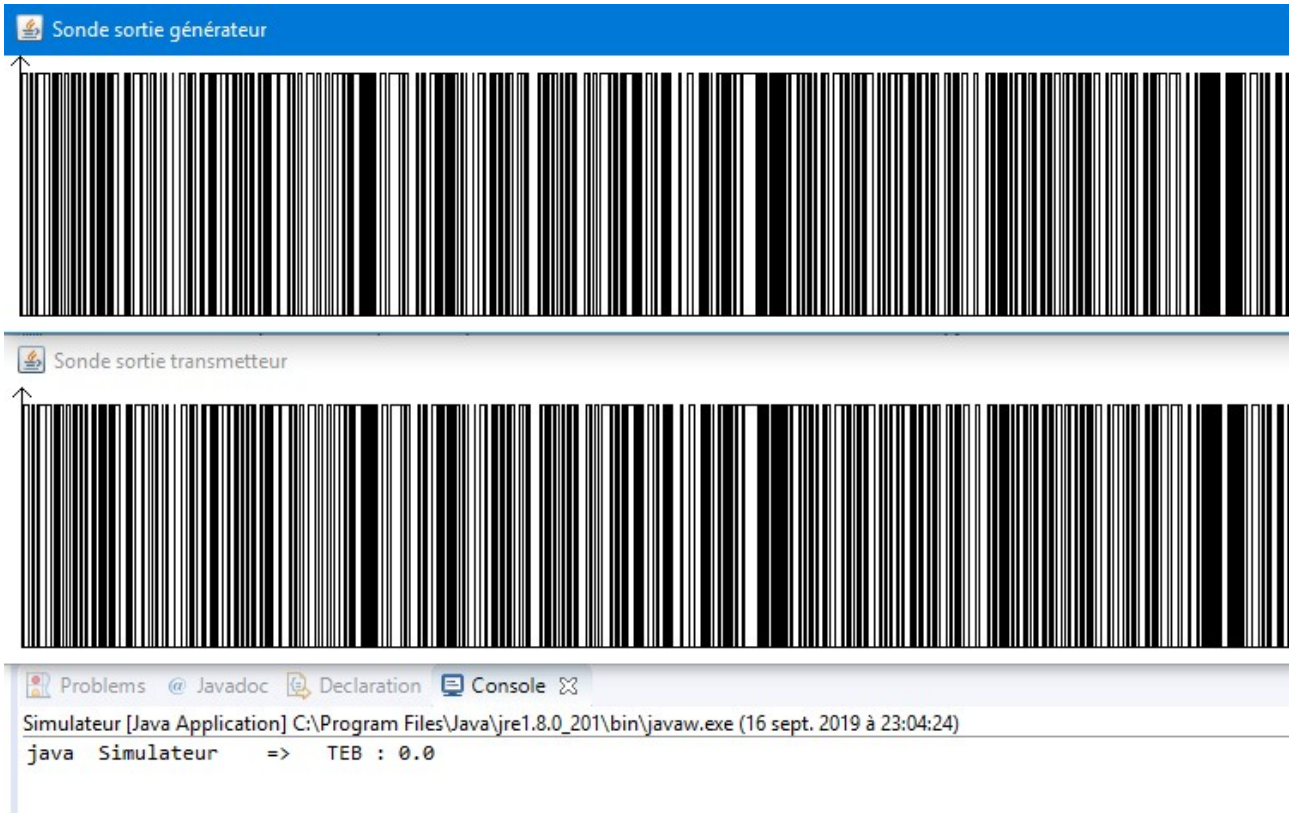

Illustration 2: Résultat transmission logique parfaite, message aléatoire

- Voici un deuxième exemple d'exécution avec l'option **-mess 11001101 -s** ; c'est à dire l'envoi d'une chaîne fixe 11001101.

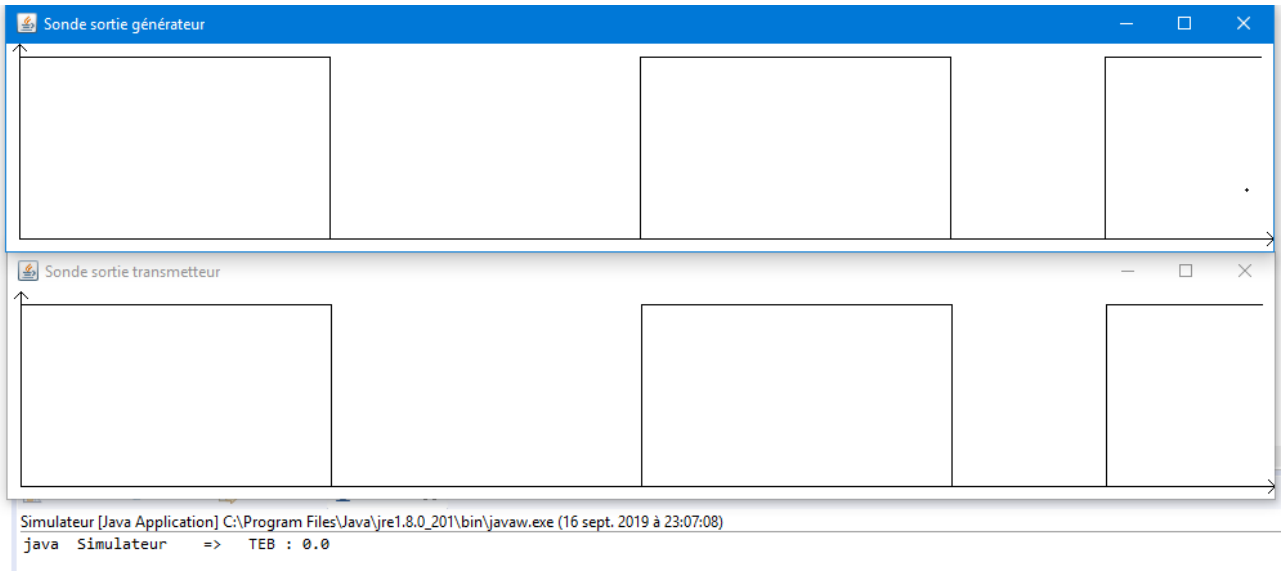

Illustration 3: Résultat transmission logique parfaite, message fixe

III.3.3 Observations

- Les tests du code ont été effectué sans classe tests, directement en manipulant le code : choix de l'efficacité, du fait du peu de classe et de leur faible complexité. La compilation a été testée sous éclipse et sous linux par un script bash ;
- Pas de gestion d'exception ajouté ;
- Les options -s et -mess sont utilisables sans problème ;
- Pas de remarque concernant les résultats obtenu par la simulation.

IV. ÉTAPE 2

IV.1 Objectif

Lors de cette deuxième étape nous travaillons en groupe de 4 avant différents objectifs :

- Réaliser une transmission non bruitée d'un signal analogique ;
- Développer des composants dans le langage de programmation Java :
  - Émetteur (NRZ / NRZT / RZ) ;
  - Récepteur (NRZ / NRZT / RZ) ;
  - Un transmetteur parfait analogique ;
- Mettre en œuvre ces composants à travers une classe de test sous la forme d'une chaîne de transmission.

IV.2 Cahier des charges

Dans cette étape nous devons mettre en œuvre un convertisseur numérique analogique et un convertisseur analogique numérique. L'objectif étant de permettre la transmission non bruité d'un signal analogique. Nous créerons ensuite un transmetteur analogique parfait capable de transmettre le signal analogique, nous vérifions cette transmission à l'aide de sondes analogiques.

Voici le modèle que nous devons construire :

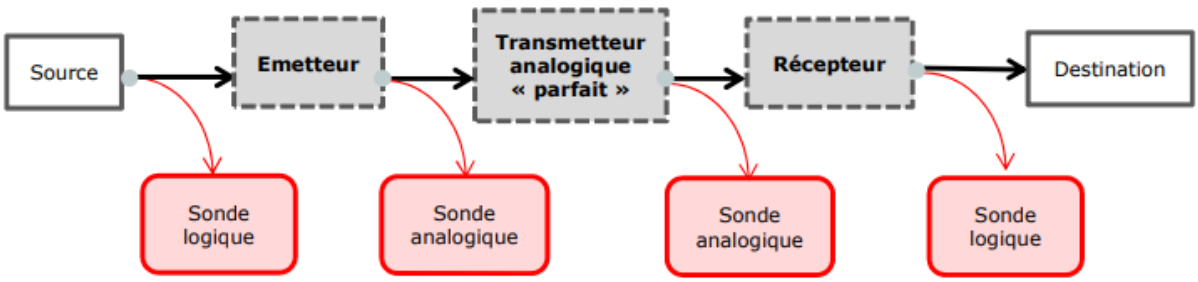

Illustration 4: Modélisation de la chaîne de transmission à l'étape 2

IV.3 Développement Logiciel

IV.3.1 Réalisation

- **Convertisseur Numérique Analogique et Analogique Numérique**

L'objectif étant de transmettre le signal numérique sur le transmetteur parfait analogique, nous allons devoir effectuer une conversion :

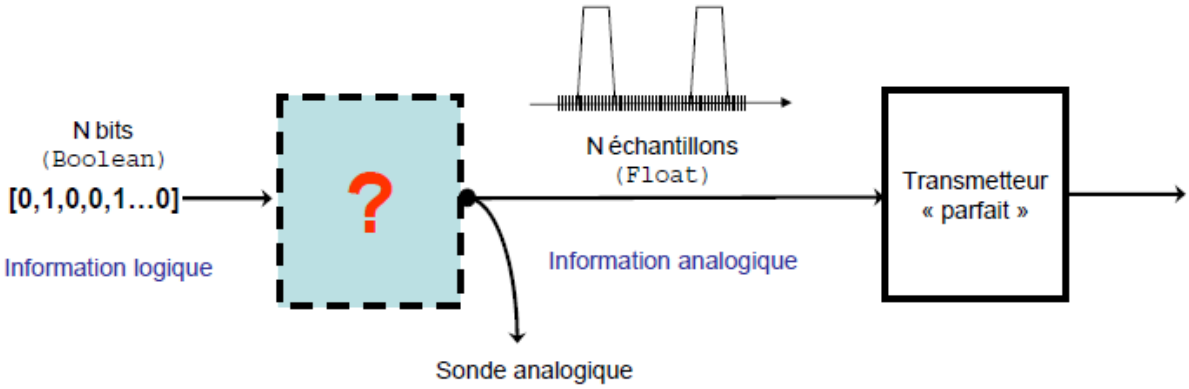

Illustration 5: Notre Système de transmission

Afin de pouvoir récupérer notre signal numérique après une transmission analogique, nous allons devoir effectuer la conversion inverse. Pour cela notre système va devoir évaluer ce qu'il reçoit, 0 ou 1. Nous accompagnerons la description de chaque codage par une explication de la méthode de conversion utilisée.

Pour se faire nous avons crée différentes classes, (cf diagramme de classes en annexe) :

- ConvertisseurAnalogiqueNumérique ;
- EmetteurNrz ;
- EmetteurNrzt ;
- EmetteurRz ;
- RecepteurNrz ;
- RecepteurNrzt ;
- RecepteurRz.

Comme ce schéma nous le précise, nous allons avoir en entrée une information logique / numérique à transmettre sous forme d'information analogique.

Pour se faire notre signal binaire sera échantillonné puis convertie en float.

Nous allons utiliser lors de cette séance trois formes de conversion numérique analogique :

- NRZ (Non Return to Zero), forme d'onde rectangulaire ;
- NRZT (Non Return to Zero), forme d'onde trapézoïdale ;
- RZ (Return to Zero), forme d'onde impulsionnelle.

(A) CODAGE NRZ

- **EmetteurNrz :**

L'émetteur Nrz reçoit un objet Information Boolean et le transforme en objet Information Float, ses objectifs :

- Récupérer l'information reçue Boolean ;
- Créer un objet Information contenant une liste de Float : chaque True devient un packet de Float de valeur amplitudeMax et de taille nbEchantillon, chaque False devient un packet de Float de valeur amplitudeMin et de taille nbEchantillon.

- **RécepteurNrz :**

Le récepteur Nrz reçoit un objet Information Float et le transforme en objet Information Boolean, ses objectifs :

- Récupérer l'information reçue Float ;
- Créer un objet Information contenant une liste de Float : chaque paquet de taille nbEchantillon dont les bits sont à amplitudemax devient un True, chaque paquet de taille nbEchantillon dont les bits sont à amplitudeMin devient un False.

- **Méthode de conversion :**

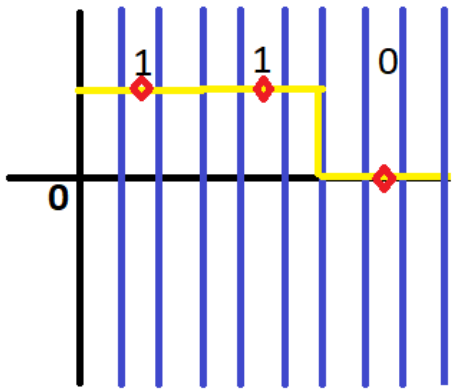

Illustration 6: Méthode de conversion NRZ

Analyse :

Sur ce schéma, les traits en bleus représentent les échantillons sur le signal reçu, ici 3 et un message de 3 bits 110.

Pour déterminer si le signa reçu est un 0 ou 1 notre système se place au niveau du losange rouge, qui correspond au centre de l'impulsion NRZ lors de la génération du signal.

(B) CODAGE NRZT

Le codage NRZT prend une forme d'onde trapézoïdale. Son temps de montée de l'amplitude min vers l'amplitude max est de un tiers du temps du bit et le temps de descente est aussi d'un tiers. Nous serons donc à un tiers du temps bit à AmplMax ou AmplMin.

Pour réaliser ce codage nous avons construit deux classes :

- **EmetteurNrzt :**

Cette classe permet d'émettre des messages avec le codage NRZT, c'est à dire des messages qui ont leur premier tiers des échantillons pour un symbole qui augmente jusqu'à la valeur maximum linéairement.

Le deuxième tiers des échantillons reste à cette valeur, puis le dernier tiers diminue linéairement jusqu'à 0 si le symbole suivant est négatif.

Pour un symbole négatif le premier tiers diminue jusqu'à la valeur minimale et le dernier tiers augmente jusqu'à 0 si le symbole qui suit est négatif.

Si des symboles consécutifs sont identiques alors la valeur entre le deuxième tiers du premier symbole et le deuxième tiers du dernier symbole consécutif la valeur reste constante.

Au sein de cette classe émetteur nous avons une méthode emettre() :

- Mise en forme de l'information reçue pour la réémettre ;
- Créer une information pour chaque symbole qui crée nbEchantillon.
- **Deux cas pour le 1er tiers :**
  - True : tend vers amplitudeMax ;
  - False : tend vers amplitudeMin ;
- **Deux cas pour le 2ème tiers :**
  - True : le symbole est constant à la valeur amplitudeMax ;
  - False : le symbole est constant à la valeur amplitudeMin.
- **Deux cas pour le 3ème tiers :**
  - Si le symbole est différent du symbole courant alors il tend vers 0 ;
  - Si le symbole est iddentique au courant alors la valeur reste constante sur le 1er tiers du suivant et sur le dernier tiers du courant.

- **RécepteurNrzt :**

Cette classe permet la réception pour des messages de type NRZT. Au sein de cette classe nous avons aussi une méthode emettre(). Cette fois, celle nous permet de récupérer l'information reçue pour comparer les valeurs des échantillons à l'aide de leurs amplitudes.

- Récupère l'information reçue ;
- Avec les nbEchantillons un tableau est rempli avec soit true ou false en comparant la valeur de l'échantillon pris en compte à l'amplitudeMax et Min ;
- Si l'échantillon > amplitudeMax/2 alors on met true dans le tableau ;
- Sinon on met false dans le tableau ;
- On crée une information avec en paramètres le constructeur du tableau, que l'on met dans informationEmise.

- **Méthode de conversion :**

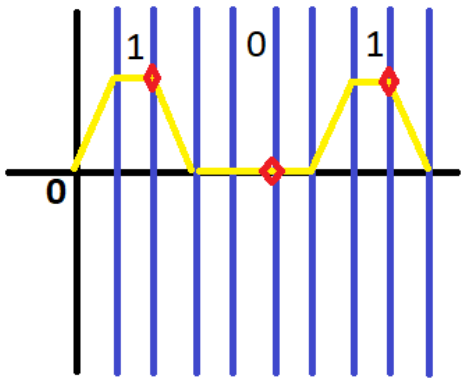

Illustration 7: Méthode de conversion NRZT

Analyse :

Sur ce schéma, les traits en bleus représentent les échantillons sur le signal reçu, ici 3 et un message de 3 bits 101.

Pour déterminer si le signa reçu est un 0 ou 1 notre système se place au niveau du losange rouge. Ce losange se situe à la fin du deuxième tiers, de cette manière le système étudie la partie gauche à partir de ce point jusqu'à la fin de la partie constante. Ainsi il peut déduire si il est sur la partie amplitudeMax ou amplitudeMin de ce signal. Ce qui lui permettra de déterminer si il s'agit d'un 0 ou 1.

(c) Codage Rz

Chaque bit est divisé en 3. De cette manière, le premier et le dernier tiers prennent l'amplitude minimale alors que sur le tiers central on a un maximum au milieu du temps du bit égal à son amplitude maximum. Ce codage est celui qui sera utilisé par défaut pour le signal analogique.

Nous avons eu à construire deux classes pour réaliser ce codage :

- **EmetteurRz :**

Pour se faire nous avons un constructeur qui contient un nombre d'échantillon ainsi que les deux amplitudes (max et min).

Au sein de cette classe et de la méthode émettre nous avons différentes variables :

nbMesure qui permet de définir le nombre de mesure en fonction du nombre d'échantillon et du nombre d'éléments contenues dans l'information reçue. Cette variable va nous permettre de définir un tableau d'informations analogiques.

Nous avons ensuite une variable tiers qui va nous permettre de deviser le tiers le nombre d'échantillons afin d'affecter les valeurs d'amplitudes max et min. Nous avons ensuite une variable tiers qui stock le nombre d'échantillons (divisé par 3) afin d'affecter les valeurs d'amplitudes max et min.

Au sein de cette méthode nous parcourons l'information reçue grâce à la variable i puis nous utilisons la variable j que nous avons initialisé à 0 pour parcourir l'échantillon et ses trois tiers. De cette manière nous allons pouvoir définir le premier tiers comme j inférieur à 1/3 comme valeur d'amplitude minimale, le second comme égal à l'amplitude maximale (si j est inférieur à 2/3) et le troisième comme amplitude minimale dans le cas ou j est inférieur à 2/3.

Enfin nous définissons cette information comme l'information à envoyer et connectons vers les destinations.

- **RecepteurRz :**

Nous souhaitons ici convertir l'information analogique reçue, pour se faire nous allons définir dans la classe RecepteurRz un constructeur, il contient les mêmes paramètres qu'émetteur Rz. Ce qui est logique puisque ce récepteur décode l'information transmise par l'émetteur Rz.

Définition de la méthode émettre :

- Émettre récupère l'information reçue sous forme de tableau de float ;
- L'information reçue est ensuite regroupée par paquet (de la taille d'un échantillon, correspondant à un bit) ;
- Chaque paquet est analysé de la manière suivante :
  - Si dans le paquet il y a une valeur correspondant à l'amplitude Max alors le paquet correspond à un bit à 1 ;
  - Sinon le paquet est un bit à 0.
- En parallèle est remplie une liste de Boolean qui sera emballée dans un objet Information puis envoyé comme Information Émise.

- **Méthode de conversion :**

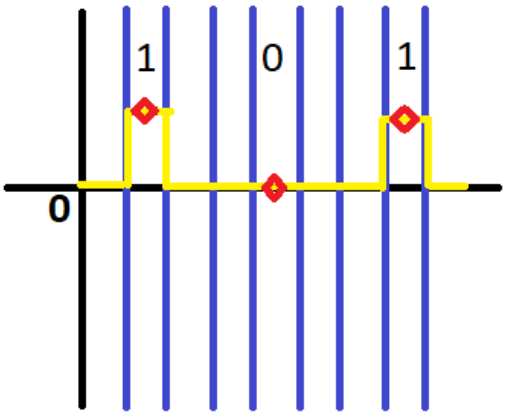

Illustration 8: Méthode conversion RZ

Analyse :

Sur ce schéma, les traits en bleus représentent les échantillons sur le signal reçu, ici 3 et un message de 3 bits 101.

Pour déterminer si le signa reçu est un 0 ou 1 notre système se place au niveau du losange rouge, qui correspond au centre de l'impulsion RZ lors de la génération du signal.

IV.3.2 Résultats

(a) Cas d'utilisation du codage RZ :

A l'aide de la classe Simulateur nous allons valider notre chaîne de transmission. Pour se faire nous utilisons les commandes uniques permettant de générer ici un message de longueur 5 sous forme RZ, d'amplitudes min = -2 et max = 2 :

```
String[] argsBis = {"-mess","5","-s","-form","RZ","-ampl","-2","2"};

try
```

Illustration 9: Simulation du codage RZ

Nous utilisons dans notre chaîne, quatre sondes, deux sondes logiques et deux sondes analogiques :

Nous pouvons voir ici notre message émis (11010) :

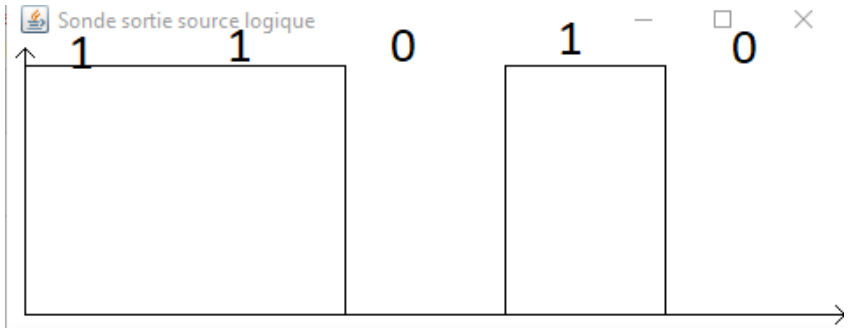

Illustration 10: RZ - Message émis 11010

Une fois ce message émis par la source logique, il va passer par l'émetteur NR, la sonde analogique connectée :

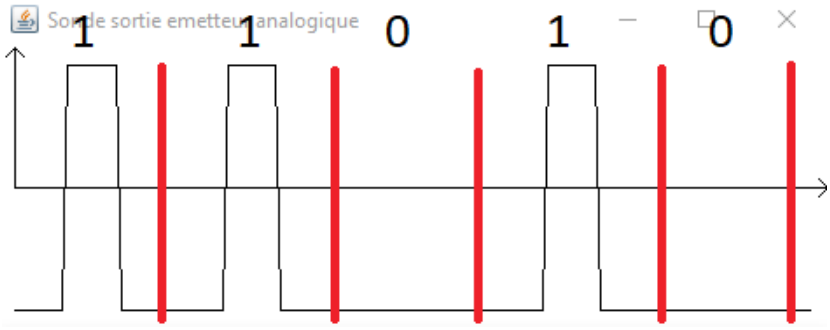

Illustration 11: RZ - Message émis en sortie de l'émetteur analogique

Ci dessus nous pouvons voir le message émis et sa conversation avant le passage dans le transmetteur analogique parfait. Chaque trait rouge permet de délimiter le temps d'un bit, nous voyons bien que notre conversion fonctionne correctement puisque pour chaque bit à 1, le signal est à l'amplitude max = 2 pendant 1 tiers du temps bit et à l'amplitude min sur 2 tiers du temps bit. Rappelons que dans cette simulation Amin = -2 et Amax = 2, par ailleurs chaque bit 0 est bien à l'amplitude min.

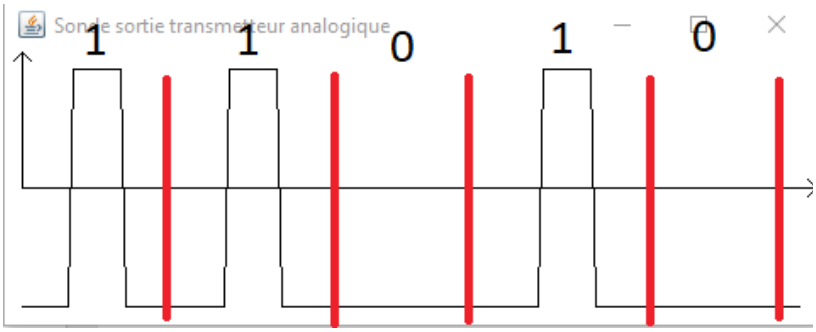

Illustration 12: RZ - Sortie transmetteur analogique

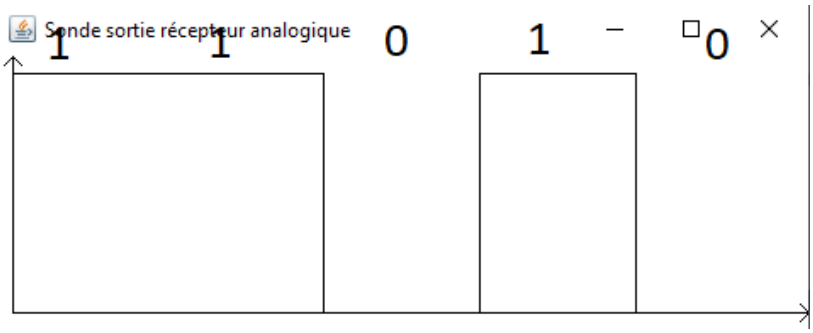

Illustration 13: RZ - Sortie récepteur analogique

Analyse :

Ci dessus nous avons le signal à la sortie du transmetteur et récepteur analogique. Nous constatons que le signal est identique à celui injecté en entrée, ainsi notre transmetteur analogique parfait fonctionne correctement. De plus le signal reçu en sortie du récepteur analogique est identique à celui émis par la source aléatoire, notre convertisseur analogique numérique fonctionne correctement aussi.

(B) CAS D'UTILISATION DU CODAGE NRZ :

A l'aide de la classe Simulateur nous allons valider notre chaîne de transmission. Pour se faire nous utilisons les commandes uniques permettant de générer ici un message de longueur 5 sous forme NRZ, d'amplitudes min = -2 et max = 2 :

```
//Test des arguments avec le String[] argBis :

String[] argsBis = {"-mess","15","-s","-form","NRZ","-ampl","-2","2"};

try
```

Illustration 14: Smulation avec le codage NRZ

Nous utilisons dans notre chaîne, quatre sondes, deux sondes logiques et deux sondes analogiques :

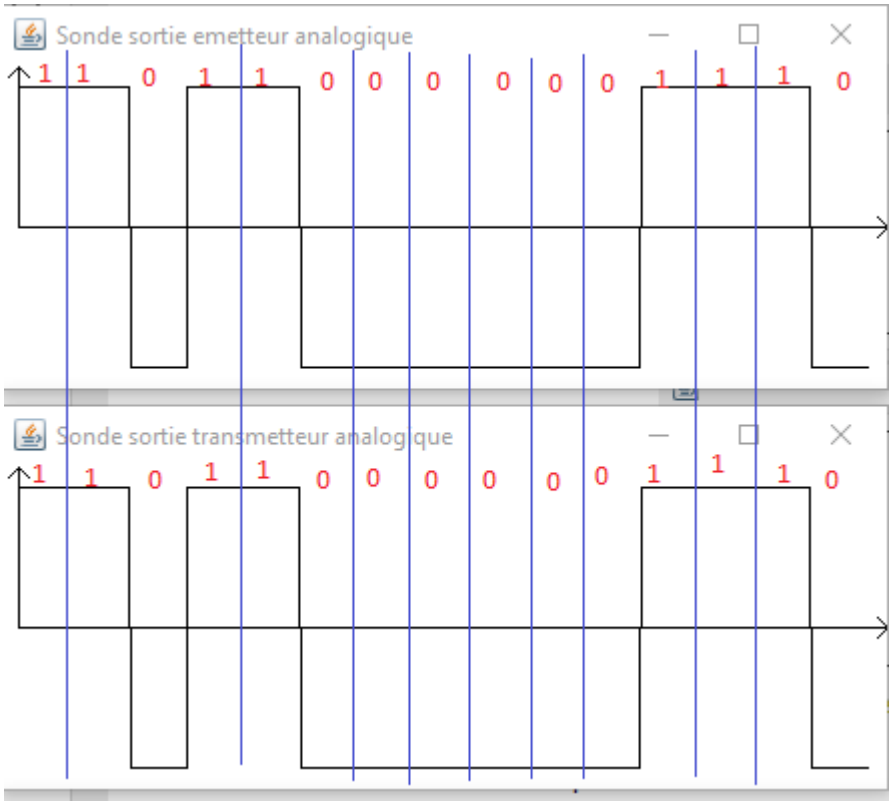

Illustration 15: NRZ - Sorties émetteur analogique et transmetteur analogique

Analyse :

Ci dessus nous pouvons observer le signal émis par l'émetteur analogique ainsi que par le transmetteur analogique parfait. Nous retrouvons bien le message de 15 bis émis par la source aléatoire avec une amplitude max de 2 et min de -2. La sortie du transmetteur analogique parfait est identique au signal reçu et notre TEB est de 0, notre chaîne de transmission fonctionne donc parfaitement. Voici le signal en sortie du récepteur analogique, qui transmet bien la chaîne de 15 bits émise par la source aléatoire :

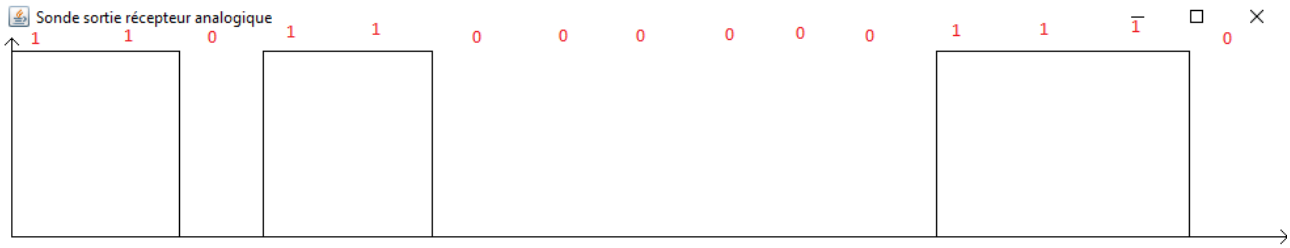

Illustration 16: NRZ - Message en sortie du récepteur analogique

Analyse :

Nous retrouvons bien la chaîne générée par le simulateur, que nous avons observé en sortie de l'émetteur analogique (1101100000001110).

(c) Cas d'utilisation du codage NRZT :

A l'aide de la classe Simulateur nous allons valider notre chaîne de transmission. Pour se faire nous utilisons les commandes uniques permettant de générer ici un message de longueur 10 sous forme NRZT, d'amplitudes min = -2 et max = 2 :

```
String[] argsBis = {"-mess","10","-s","-form","NRZT","-ampl","-2","2"};

try
{
```

Illustration 17: Simulation codage NRZT

Voici ce que nous obtenons sur les sondes en sorties de l'émetteur et du transmetteur :

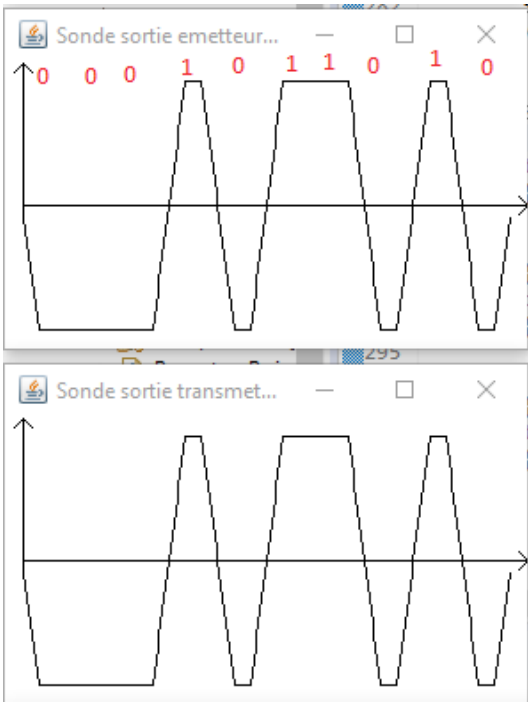

Illustration 18: NRZT - Sondes sortie émetteur et récepteur

Analyse :

Ci dessus nous pouvons observer le signal émis par l'émetteur analogique ainsi que par le transmetteur analogique parfait. Nous retrouvons bien le message de 10 bis émis par la source aléatoire avec une amplitude max de 2 et min de -2. La sortie du transmetteur analogique parfait est identique au signal reçu et notre TEB est de 0, notre chaîne de transmission fonctionne donc parfaitement. Voici le signal en sortie du récepteur analogique, qui transmet bien la chaîne de 10 bits émise par la source aléatoire :

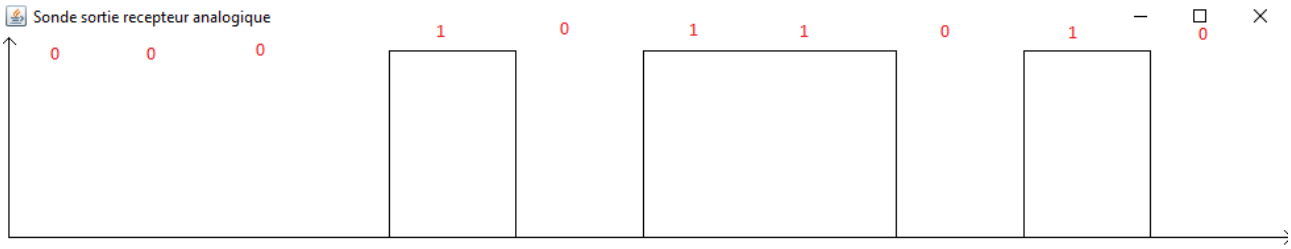

Illustration 19: NRZT - Signal en sortie du récepteur analogique

IV.4 Observations

L'objectif de cette étape 2 était crée une chaîne de transmission non bruitée d'un signal analogique avec différents codages (NRZ, NRZT, RZ). Nous avons réussi à réaliser cette chaîne tout en conservant le caractère parfait de notre transmetteur analogique.

V. ÉTAPE 3

V.1 Objectifs

- Réaliser une chaîne de transmission non-idéale avec un canal bruité de type « gaussien » ;
- Développement d'un composant capable de générer un bruit blanc gaussien ;
- Modifier nos composants pour qu'ils prennent en compte le bruit et puissent transmettre un message correcte.

V.2 Cahier des charges

Dans cette étape nous allons devoir mettre en œuvre un transmetteur bruité analogique de type gaussien. La propagation dans le canal est modélisée par un bruit blanc additif gaussien. Voici le schéma de la chaîne de transmission que nous allons construire :

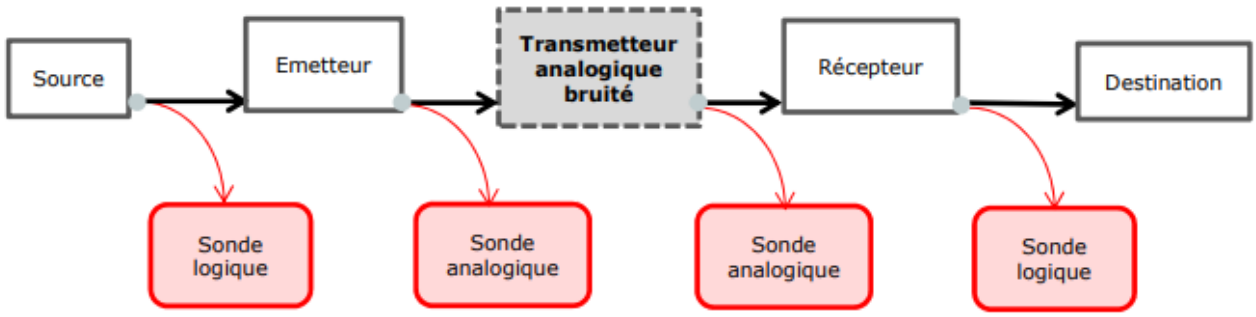

Illustration 20: Chaîne de transmission étape 3

Le bruit blanc gaussien est un processus aléatoire dont la densité spectrale de puissance est constante quelle que soit la fréquence. Ce processus aléatoire suit une loi normale caractérisée par une variance et une moyenne.

V.3 Développement logiciel

V.3.1 Transmetteur Analogique Bruité

Voici la spécification de notre classe transmetteur analogique bruité. Au sein de cette classe nous avons crée différentes méthodes et fonctions qui vont nous permettre d'introduire un bruit blanc gaussien au sein de cette chaîne de transmission.

Nos méthodes :

- **calculPuissance :** cette méthode prend en paramètres nbEchantillon et informationRecue, elle va calculer la somme des échantillons (au carrés) divisé par le nombre d'échantillons :

  ⇨ Calcul de la puissance du signal

  $$P_s = \frac{\mathbb{E}(|a_k|^2)}{N} \sum_{n=1}^{N} h^2(n) = \lim_{K \to +\infty} \frac{1}{K} \sum_{n=1}^{K} s^2(n) \quad \begin{matrix} a_k\text{: symb. transmis} \\ h(n)\text{: filtre de mise en forme} \end{matrix}$$

- **calculSigma :** cette méthode permet de calculer le sigma, pour déduire ce sigma on va utiliser la puissance de bruit :

  - (On a puissanceBruit = Ps / (10^(SNR/10)))
  - sigma = sqrt(puissanceBruit)

- **generationBruitBlanc :** cette méthode va nous permettre de générer un bruit blanc, pour se faire nous allons utiliser deux variables a1 et a2, qui suivent des lois uniformes :

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

- **ajoutSignalBruite :** cette méthode nous permet d'ajouter le signal bruité à l'aide d'une liste que nous avons crée.
- **Fonction d'export :** cette méthode d'export nous permet d'exporter les valeurs d'un signal dans un fichier texte, ce qui va nous permettre de réaliser l'histogramme.

Nous allons utiliser ces différentes méthodes, pour dans un premier temps calculer la puissance du signal. Nous convertissons ensuite cette puissance en dB. Une fois cette conversion faite nous allons calculer le SNR.

Pour le calcul de la gaussienne par rapport au bruit nous avons réalisé un histogramme, pour cela nous avons trié les valeurs récupérées par ordre croissant, puis dans l'histogramme nous les avons adaptés à des tranches de valeurs :

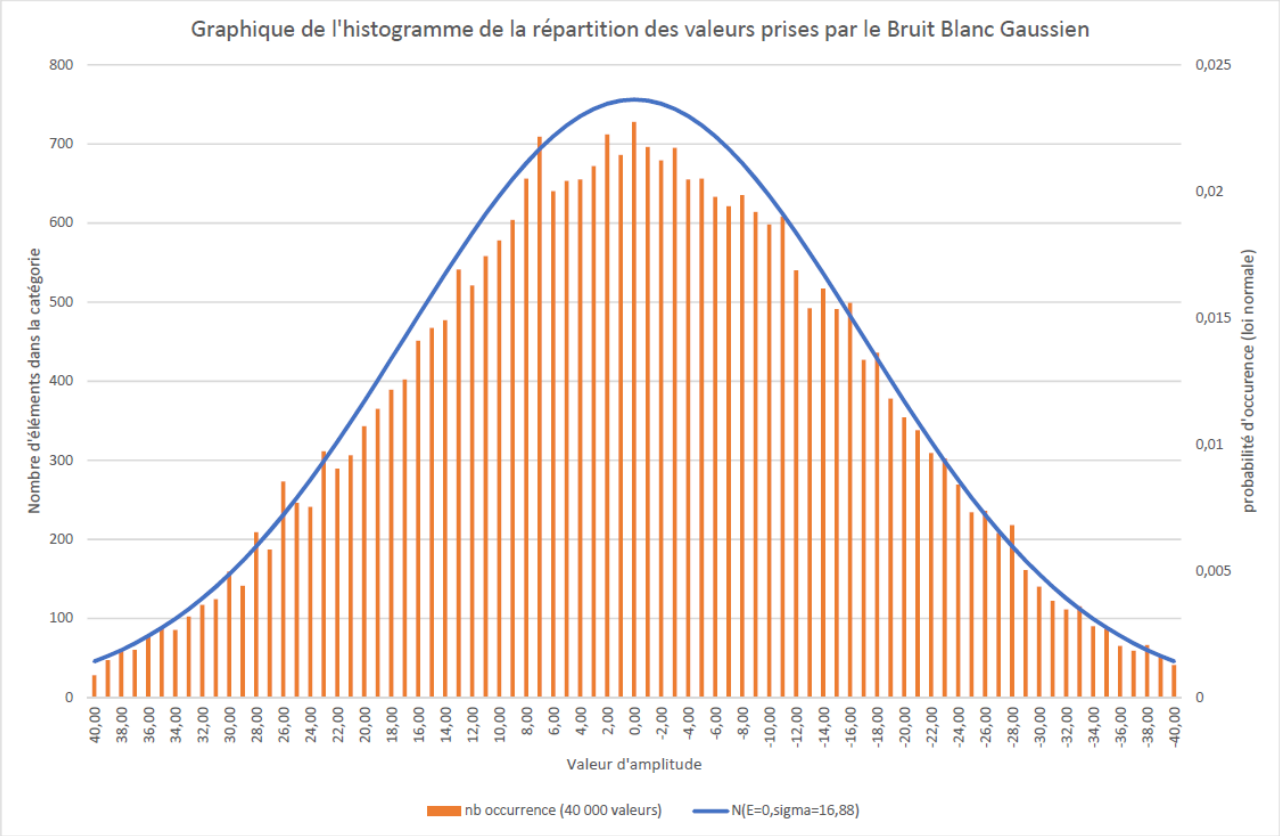

Illustration 21: Graphique de l'histogramme de la répartition des valeurs prises par le Bruit Blanc Gaussien

V.3.2 Étude théorique

Concernant cette chaîne de transmission, nous attendons que pour chaque message transmission le transmetteur analogique introduit un bruit de type gaussien généré à l'aide du SNR que nous avons mis en paramètre. A la fin de la chaîne de transmission, les signaux décodés pourront comporter des erreurs suivant le SNR que nous avons mis en paramètre, tout en sachant que plus celui-ci est élevé plus notre signal en sortie sera identique au signal émis.

V.3.3 Résultats

Au sein de nos résultats nous allons vous montrer comment chacun de nos codage (NRZ, NRZT et RZ) fonctionnent une fois passé dans le transmetteur bruité avec les paramètres suivants :

- **15 bits à envoyer**
- **AmplitudeMin=-2**
- **AmplitudeMax= 2**
- **SNR 2db**

Dans l'ordre voici les trois codages :

- **Codage NRZ :**

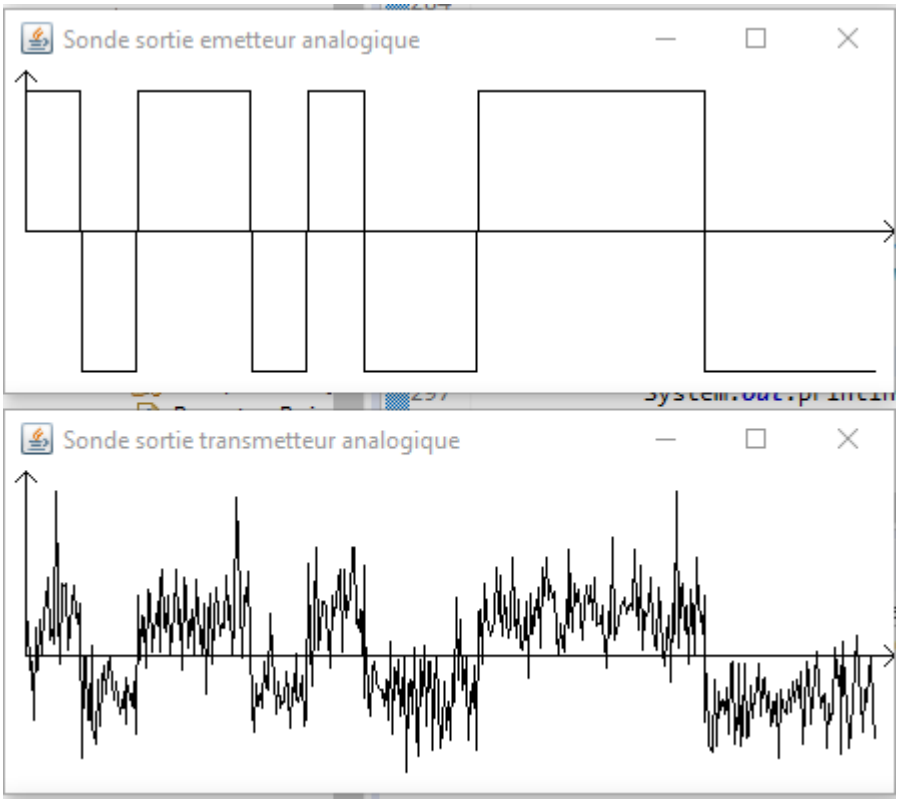

Illustration 22: NRZ - Sondes sorties émetteur et transmetteur analogique

- **Codage RZ :**

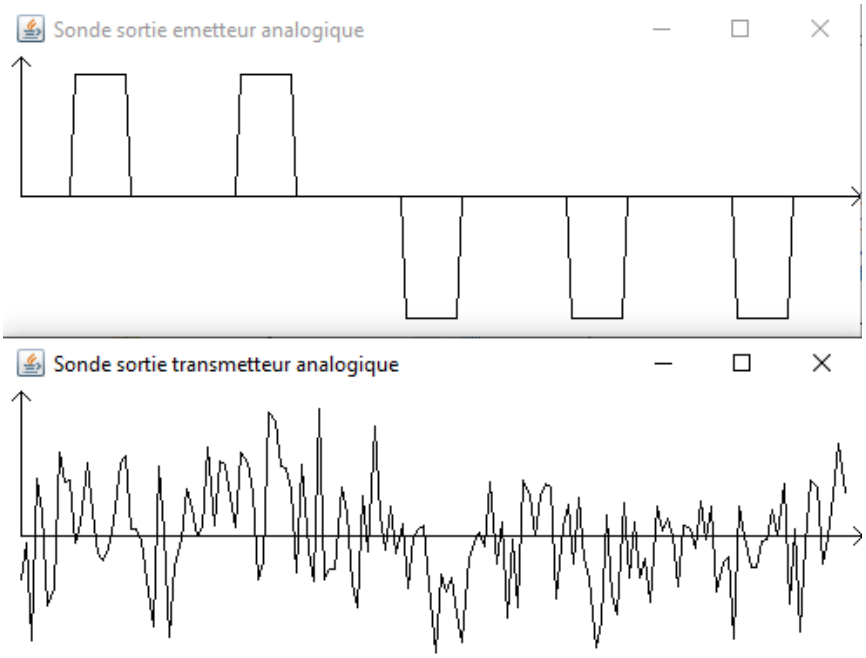

Illustration 23: RZ - Sondes sorties émetteur et transmetteur analogique

- **Codage NRZT :**

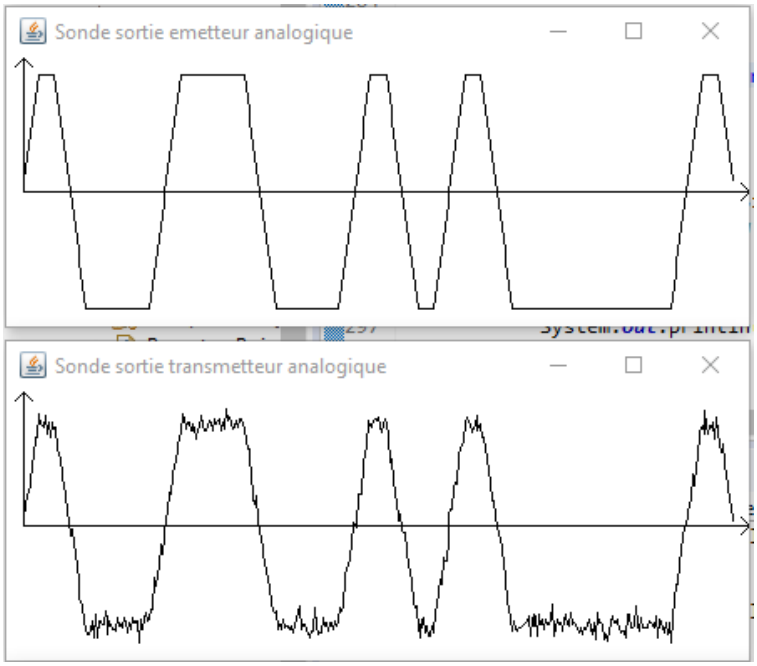

Illustration 24: NRZT - Sondes sorties émetteur et transmetteur analogique

De plus nous pouvons constater sur les sondes suivantes qu'après ajout d'un bruit blanc gaussien par notre transmetteur nous récupérons bien l'information émise, avec ici le codage RZ d'utilisé :

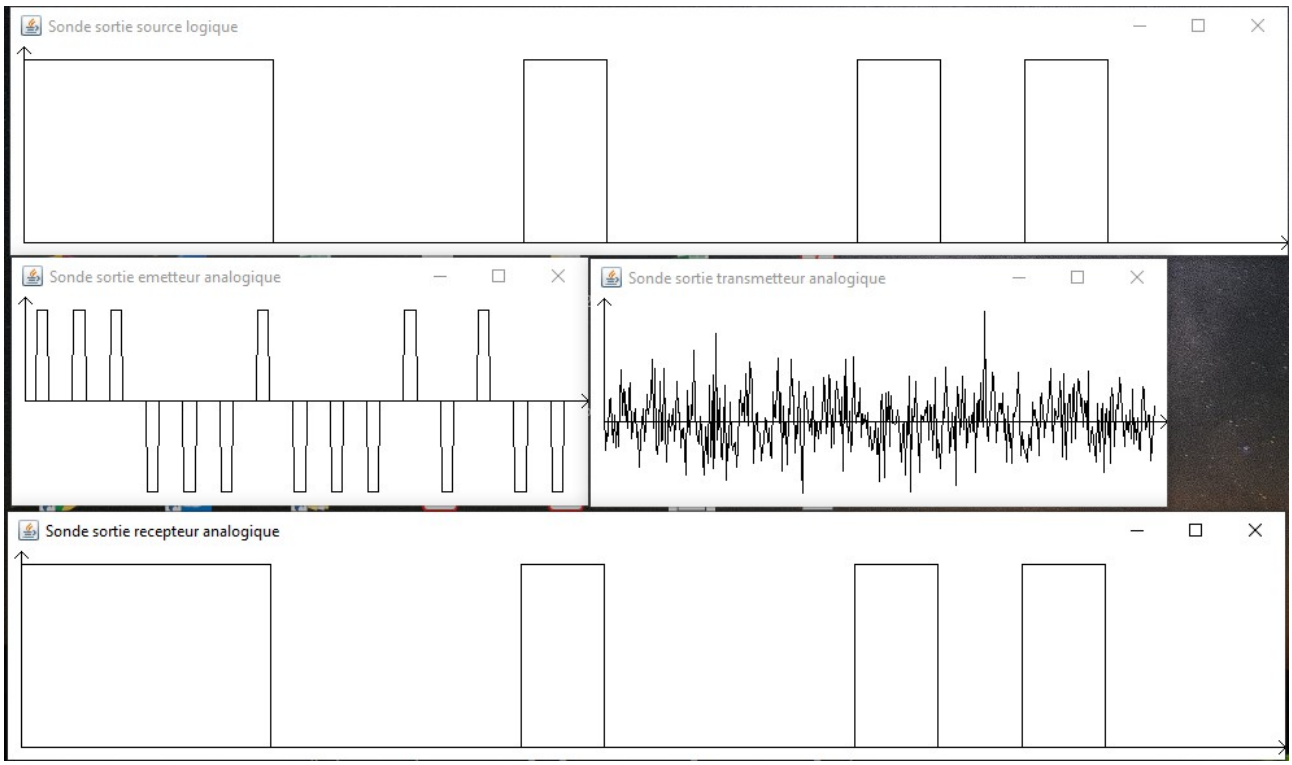

Illustration 25: RZ - Comparaison signal émis / signal reçu avec une transmission bruitée

Nous voyons bien que ces 3 signaux sont bruités après le passage dans notre transmetteur analogique bruité.

(A) Résultats et comparaisons avec la Probabilité d'erreur binaire d'un NRZ

Dans cette partie nous allons comparer les valeurs du TEB pour les codages RZ, NRZ et NRZT. Puis nous comparerons les valeurs du TEB du codage NRZ avec la probabilité théorique d'erreur binaire d'un NRZ bipolaire. Cette valeur théorique sera notre courbe de référence. Pour se faire nous allons simuler une transmission de 1000 bits, avec un TEB pour ces différentes simulations allant de 0 à 10.

Voici un exemple de l'une des simulations que nous avons réalise :

```
String[] argsBis = {"-s","-mess","1000","-form","RZ","-ampl","-2","2","-snr","1"};
//,"-ti","2","0.01"
```

Illustration 26: Simulation RZ avec 1000 bits

Les différentes simulations nous ont permis de construire le tableau suivant :

	TEB		
SNR	NRZ	RZ	NRZT
0	0,084	0,049	0,161
1	0,057	0,043	0,0865
2	0,038	0,028	0,1175
3	0,026	0,019	0,04
4	0,014	0,007	0,0315
5	0,008	0,004	0,019
6	0,005	0,002	0,023
7	0	0	0,0045
8	0	0	0,008
9	0	0	0,0045
10	0	0	0,0015

Illustration 27: Valeurs du TEB - NRZ - RZ - NRZT

En plus de ces valeurs de TEB nous avons utilisé le taux d'erreur binaire comme courbe de référence. Pour tracer cette courbe nous avons utilisé la formule suivante, qui est la probabilité d'erreur binaire :

$$P_b = \frac{1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right).$$

Ce qui nous a permis d'établir le tableau suivant :

EbNo_db	EbNo	Proba erreur NRZ Bipolaire
0	1	0,078649604
1	1,25892541	0,037506128
2	1,58489319	0,012500818
3	1,99526231	0,002388291
4	2,51188643	0,000190908
5	3,16227766	3,87211E-06
6	3,98107171	9,00601E-09
7	5,01187234	6,81019E-13
8	6,30957344	2,2674E-19
9	7,94328235	1,39601E-29
10	10	1,04424E-45

Illustration 28: Valeurs de la Probabilité théorique d'erreur d'un NRZ Bipolaire

Avec EbNo = 10^(EbNo_db/10).

Ces résultats nous ont permis de tracer la courbe de référence pour le NRZ bipolaire, sur ce graphe nous avons ajouté nos valeurs de NRZ pour tracer notre courbe de simulation :

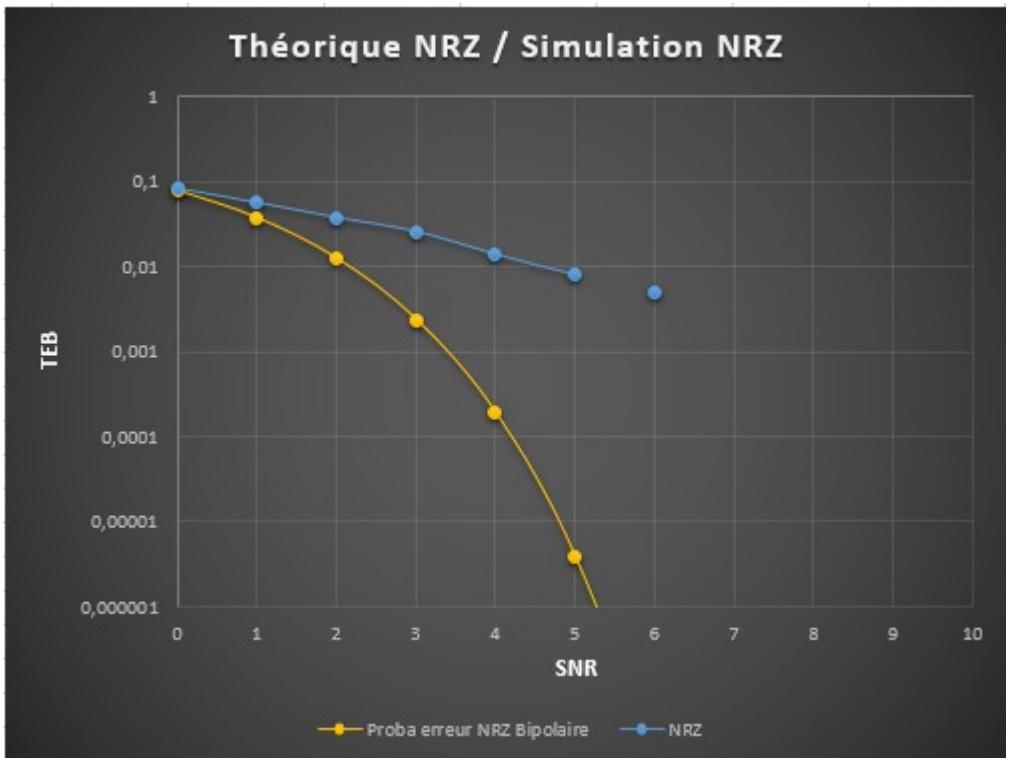

Illustration 29: Comparaison NRZ théorique et simulation

Analyse :

Sur ce graphique nous pouvons voir en jaune la courbe de référence de la probabilité d'erreur binaire théorique NRZ et en bleu la courbe NRZ que nous obtenons en faisant évoluer le rapport Eb/No. Nous constatons que lorsqu'il n'y a pas de bruit dans notre chaîne de transmission le TEB est de 10^-1 comme pour la courbe de référence. Cependant une fois que nous injectons du bruit dans notre système nous nous éloignons des valeurs théoriques pour le NRZ. Par exemple pour un SNR égal à 4 nous obtenons un TEB proche de 10^-2 contre 10^-4 selon la courbe de référence. Ce résultat et la comparaison de ces deux courbes nous permettent de constater qu'un système ne sera jamais identique à une courbe de référence, l'objectif n'étant pas d'obtenir les mêmes valeurs mais d'avoir un système. Par ailleurs, pour un SNR de plus de 7 nous obtenons un TEB égal à 0.

(B) COMPARAISONS NRZ / RZ / NRZT

En plus de comparer le TEB du NRZ avec la probabilité d'erreur binaire du NRZ Bipolaire nous avons comparé les différents codages entre eux :

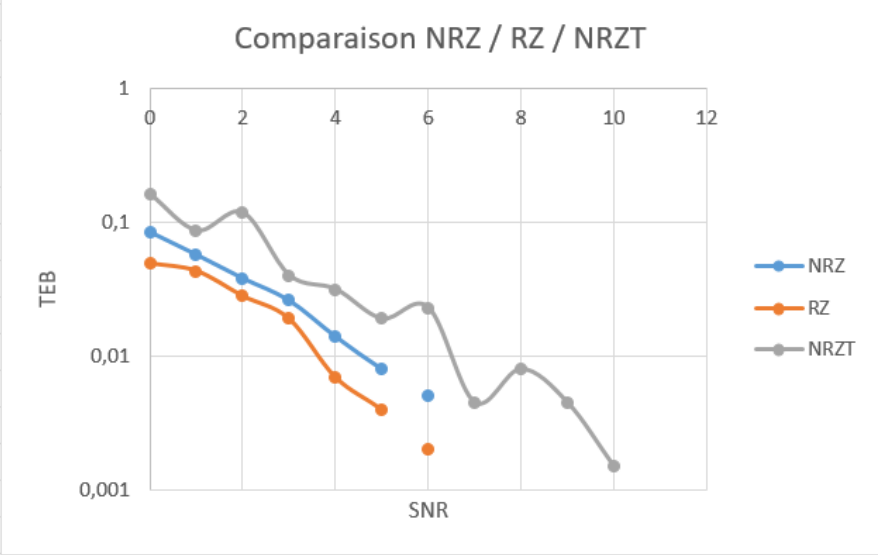

Illustration 30: Comparaison du TEB NRZ / RZ / NRZT

Analyse :

Sur ce graphique nous pouvons distinguer l'évolution du TEB en fonction du SNR en fonction des différents codages NRZ, RZ et NRZT. Nous pouvons constater que ces différents codages ont tous une valeur pour un SNR égal à 0 proche de 10^-1, ensuite l'évolution du SNR nous permet d'observer des similitudes entre les codages RZ et NRZ. Comme lorsque nous avons comparé le NRZ à la probabilité d'erreur binaire du NRZ bipolaire nous constatons qu'à partir d'un SNR de 6 nous atteignons un TEB de 0. Le NRZT quant à lui approche pour un SNR de 10 une valeur de TEB d'ordre 10^-3. Le NRZT et le RZ ont le même récepteur, cependant nous observons une différence dans l'évolution de leurs TEB, ces différences s'expliquent par la méthode de codage qui diffère.

VI. ÉTAPE 4A

VI.1 OBJECTIFS

- Réaliser une transmission non-idéale avec divers bruits réels ;
- Créer des bruits réels au travers d'un nouveau transmetteur ;
- Insérer des modèles physiques de bruit ;

VI.2 CAHIER DES CHARGES

Lors de cette étape 4A nous allons modifier notre chaîne de transmission afin d'introduire des bruits réels. Pour ajouter ces perturbations au niveau du transmetteur nous allons développer en Java un nouveau Transmetteur. La perturbation que nous allons ajouter suivra le modèle physique des trajets-multiples en suivant le modèle ci dessous :

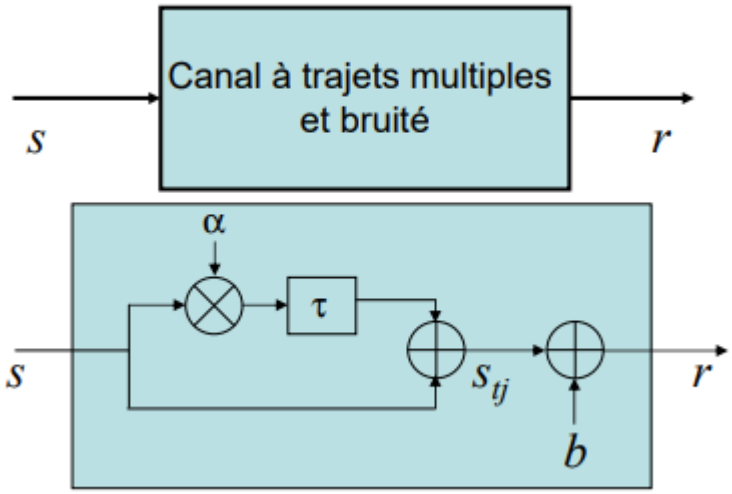

Illustration 31: Schéma trajets-multiples

Nous recevons un signal de la forme :

$$r(t) = s(t) + \alpha s(t-\tau) + b(t)$$

- b(t) est un Bruit blanc gaussien centré ;
- α représente l'atténuation du second trajet, cette atténuation est comprise entre 0 et 1 ;
- T est le retard du second trajet, qui correspond au nombre d'échantillons.

VI.3 Développement logiciel

VI.3.1 Transmetteur Analogique Multi trajets

Nous avons choisi comme vous pourrez le voir dans le diagramme de classes de l'étape 4A en annexe, de créer une classe Transmetteur Analogique Multi-trajets qui va nous permettre à l'aide de différentes méthodes de créer divers trajets bruitant notre signal. La méthode la plus importante de cette classe est celle qui nous permet de créer plusieurs trajets :

- CreationMultiTrajets :

Nous avons crée plusieurs variables :

- **informationMultitrajet** : variable qui permet de de retourner l'information, de type Information ;
- **informationTrajet** : variable temporaire qui permet de stocker un trajet ;
- **informatiionMultitrajetList** : liked list permettant de stocker tous les informationTrajet ;
- **nbTrajet :** varibale permettant de compter le nombre de trajets à effectuer.

Nous avons ensuite au sein de cette classe mis une boucle qui permet tant qu'il reste des trajet à effectuer d'instancier un nouveau trajet. Pour crée un trajet décalé on ajoute des 0 au début du nouvel objet Information. Puis on ajoute les éléments de l'informationRecue intialement en enlevant les éléments finaux. De plus on change l'amplitude en fonction du trajet des valeurs de l'informationRecue.

L'information de chacun de nos trajets est stockée dans la likedList que nous avons crée à cet effet, puis on passe au trajet suivant.

Une fois tous les trajets stockés dans la likedList d'Informations informationMultitrajetsList on additionne chaque éléments à son emplacement dans l'Information retournée que nous avons appelé informationMultitrajet.

VI.3.2 Étude théorique

En simulant cette chaîne de transmission nous espérons pour chaque message transmis introduire un bruit de type gaussien, que nous allons générer à partir du SNR (introduit lors de l'étape 3) et des trajets multiples. A la fin de la chaîne de transmission, nos signaux décodés en sorties des récepteurs comporteront des erreurs dues au SNR, ce que nous avons déjà pu observer lors de l'étape 3 à l'aide du TEB. Ces erreurs seront aussi dues aux trajets multiples que nous allons générer.

VI.3.3 Résultats

(a) Différents scénarios

Pour visualiser le fonctionnement de cette nouvelle chaîne de transmission et le décalage du au multi-trajets nous allons réaliser une simulation en codage NRZT selon plusieurs critères, sans multi-trajets, avec multi-trajets et cela avec différents paramètres identiques (SNR, Message, Nombre d'échantillons, bruit gaussien).

Voici les paramètres de notre simulation :

```
String[] argsBis = {"-s","-mess","2000","-form","NRZT","-ampl","-2","2","-snr","6"};
```

Illustration 32: Paramètres simulation étape 4

Ces paramètres vont nous permettre de simuler une chaine de transmission avec l'émission de 2000 bits, en utilisant le codage NRZT, d'amplitude max 2 et min -2 avec un SNR de 6.

- 1ère simulation sans multi-trajets :

```
 AmplitudeMax : 2.0 AmplitudeMin : -2.0 SNR : 6.0 Encodage : NRZT
java  Simulateur    =>  TEB : 0.008999999612569809
```

- 2ème simulation avec un multi-trajets (décalage de 5 et amplitude relative du signal de 2) :

```
 AmplitudeMax : 2.0 AmplitudeMin : -2.0 SNR : 6.0 Encodage : NRZT
java  Simulateur    =>  TEB : 0.03700000047683716
```

- 3ème simulation avec multi-trajets (décalage de 20 et et amplitude relative du signal de 2) :

```
 AmplitudeMax : 2.0 AmplitudeMin : -2.0 SNR : 6.0 Encodage : NRZT
java  Simulateur    =>  TEB : 0.28200000524520874
```

Analyse :

Nous avons simulé trois scénarios différents, l'objectif de ces scénarios étant d'observer l'impact du multi-trajets sur le taux d'erreur binaire et donc sur la qualité du message reçu. Ces observations nous on permis de mettre en évidence l'impact d'un décalage temporel en nombre d'échantillons entre le trajet indirect du signal et le trajet direct. Nous avons dans un premier temps introduit un décalage temporel de 5 échantillons, ce qui a fait évoluer le TEB de 10^-3 à 10^-2. Notre dernier scénario consistait à décaler de 20 échantillons, ce qui nous a donné un TEB pour une même simulation de 10^-1.

L'impact du multi-trajets sur le taux d'erreur binaire est donc bien observé, nous constatons une augmentation importante du TEB qui pour un décalage de 20 échantillons amène presque 20 % d'erreurs supplémentaires.

(B) COMPARAISON DES DIFFÉRENTS CODAGES AVEC LE MULTI-TRAJETS

Afin de compléter notre analyse d'un scénario mettant en évidence l'impact du multi-trajets sur notre chaîne de transmission, nous avons réalisé différentes simulations pour comparer l'impact du multi-trajets sur nos différents codages.

Pour se faire nous avons réalisé deux simulations en ne faisant varier que le SNR de 0 à 10 pour chacune d'entre elles :

- 1ère simulation :

```
String[] argsBis = {"-s","-mess","1000","-form","NRZT","-ampl","-2","2","-snr","10", "-ti","10","2"};
```

- 2ème simulation :

```
String[] argsBis = {"-s","-mess","1000","-form","NRZ","-ampl","-2","2","-snr","1", "-ti", "20", "4"};
try
```

Ces deux simulations nous ont permis d'établir le tableau de valeurs suivant :

	TEB avec -ti, 10, 2			TEB avec -ti, 20, 4		
SNR	NRZ10	RZ10	NRZT10	NRZ20	Rz20	NRZT20
0	0,141	0,236	0,204	0,312	0,365	0,37
1	0,097	0,192	0,177	0,305	0,328	0,381
2	0,082	0,187	0,151	0,282	0,321	0,349
3	0,082	0,148	0,142	0,29	0,323	0,344
4	0,0689	0,1	0,123	0,28	0,305	0,347
5	0,0549	0,093	0,096	0,296	0,277	0,337
6	0,035	0,0729	0,07	0,309	0,252	0,342
7	0,026	0,048	0,046	0,335	0,238	0,349
8	0,017	0,026	0,037	0,317	0,199	0,367
9	0,007	0,0149	0,019	0,335	0,187	0,372
10	0,004	0,008	0,014	0,333	0,163	0,386

A l'aide de ces valeurs nous avons tracé différentes courbes permettant de mettre en évidence l'impact de l'augmentation du décalage temporel sur le taux d'erreur binaire de chaque codage :

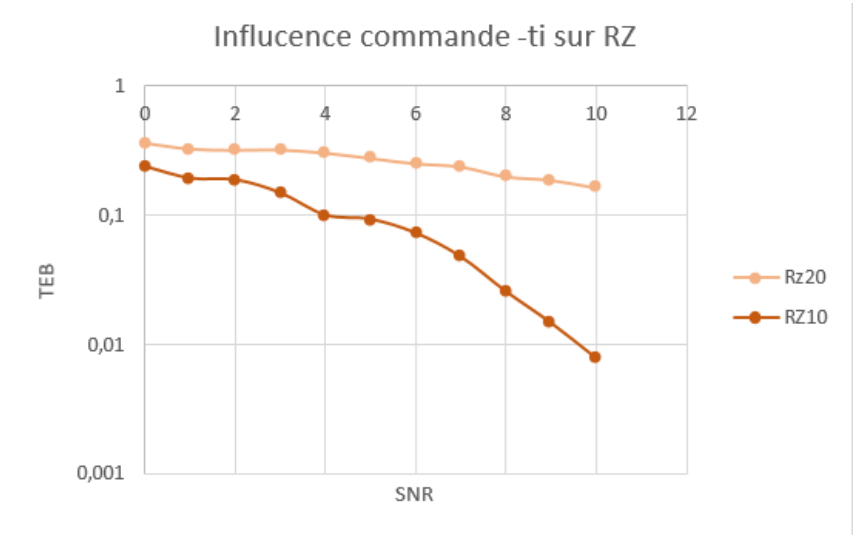

Illustration 33: RZ - Influence du décalage temporel

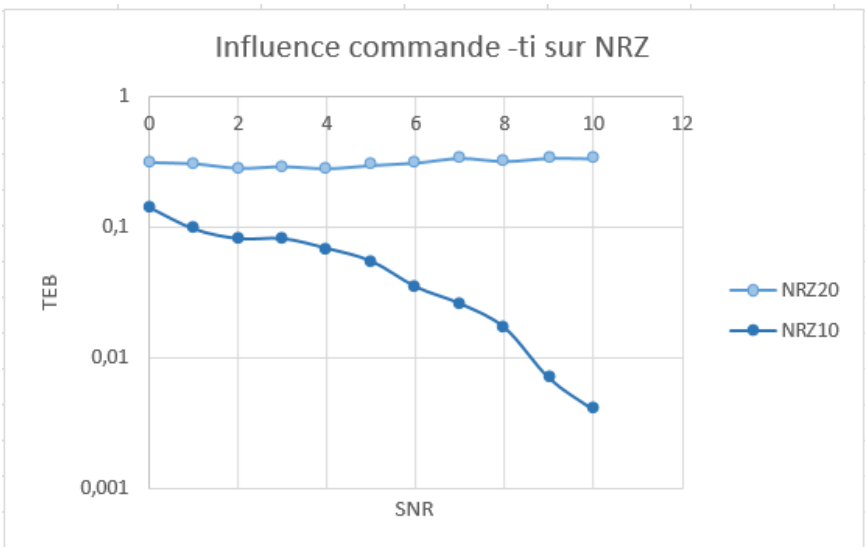

Illustration 34: NRZ - Influence du décalage temporel

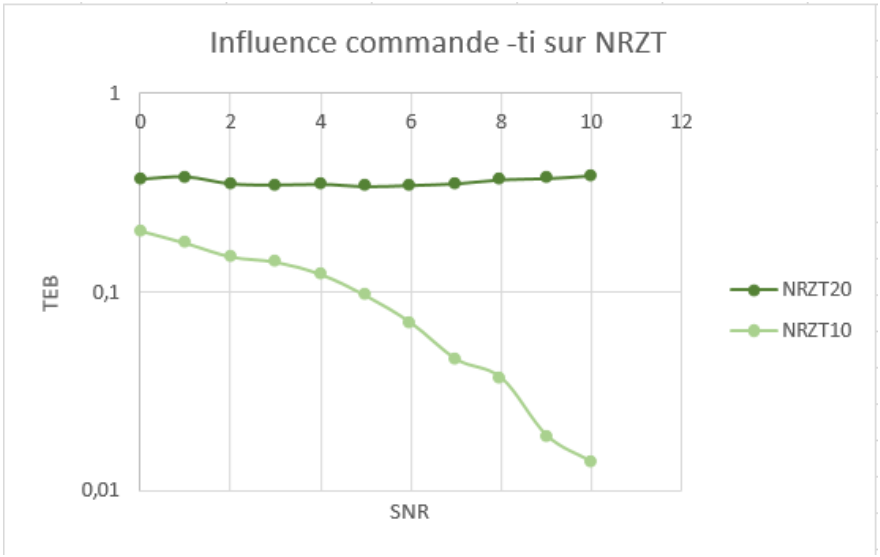

Illustration 35: NRZT - Influence du décalage temporel

Analyse :

Ces différentes courbes accompagnent notre simulation de scénarios dans la démonstration de l'impact du multi-trajets sur l'augmentation du taux d'erreurs binaires à la réception du message.

Nous pouvons constater dans chacun des cas qu'en augmentant le décalage temporel de 10 à 20 échantillons le TEB est plus important.

VII. OBSERVATIONS

Après avoir observé et simuler différents cas de multi-trajets en utilisant nos différents codages NRZ, NRZT et RZ tout en jouant avec les paramètres, d'amplitude, nombre d'échantillons etc.. Nous réalisons que le bruit que nous créons en essayant de reproduire un environnement physique et des multi-trajets a un impact sur la réception de notre message dans son intégralité. En effet nous avons constaté que plus le décalage temporel augmentait plus le TEB se détériorait. Cette conclusion nous permet de valider l'objectif de simuler un environnement physique réel tel que nous pouvons le rencontrer dans notre utilisation quotidienne des différents systèmes de transmission que nous utilisons. Maintenant que nous avons réussi à introduire ce principe de multi-trajets, nous constatons que notre système est sensible à ce phénomène, nous allons donc devoir améliorer notre chaîne de transmission pour diminuer le TEB même en présence de phénomène physiques réels.

VIII. ÉTAPE 5

VIII.1 Objectifs

- Corriger les erreurs signalés à l'étape précédente
- Intégrer le codage canal dans notre chaîne de transmission
- Mesurer l'intérêt d'un tel procédé avec notamment des comparaisons
- Mettre en évidence l'amélioration de synchronisation émetteur / récepteur

VIII.2 Cahier des charges

Cette dernière étape va nous permettre d'introduire un codage de source. L'objectif de ce codage est de diminuer le TEB, Taux d'Erreur Binaire de notre chaîne de transmission bruitée. Pour se faire nous allons insérer ce codage après la source et avant la destination. Le dispositif d'émission transformera le signal logique émis par la source en repérant et réparant les erreurs issues de la transmission bruitée avant la réception du signal par le dispositif de réception. Le repérage et la réparation des erreurs se fera par un encodeur qui transformera un bit par une séquence de trois bits.

VIII.3 Développement logiciel

VIII.3.1 Étude théorique

Dans cette dernière partie nous avons crée la classe transmetteur codage (cf diagramme de classes étape 5). Cette classe nous permet de créer la redondance sur le signal émis. Et ceci grâce à la méthode recevoir (), qui va récupérer le signal envoyé par l'émetteur pour le traiter. Le traitement de ce signal consiste à créer une information dans laquelle nous allons mettre par exemple 3 bits « 101 » pour un bit « 1 » reçu. L'information crée est ensuite envoyée vers les destinations.

Nous avons au sein de cette classe plusieurs méthodes :

- Recevoir : récupère l'information envoyée.
- Emettre : méthode qui va insérer de la redondance dans l'information pour ensuite la réémettre. Il s'agit de la méthode expliqué dans le fonctionnement de cette classe précédemment.

Notre décodeur que nous avons réalisé avec la nouvelle classe recepteurCodeur se charge de transformer les séquences de 3 bits reçus en un bit à transmetteur de la façon suivante :

$$0\,\mathbf{0}\,0 \rightarrow 0$$

$$\mathbf{0}\,0\,1 \rightarrow 1$$

$$0\,1\,0 \rightarrow 0$$

$$0\,1\,\mathbf{1} \rightarrow 0$$

$$1\,0\,\mathbf{0} \rightarrow 1$$

$$1\,0\,1 \rightarrow 1$$

$$\mathbf{1}\,1\,0 \rightarrow 0$$

Illustration 36: Méthode de décodage des séquences 3 bits

VIII.3.2 Résultats

Nous allons dans cette partie chercher à mettre en évidence l'amélioration apportée par ce décodeur pour le TEB en fonction des principaux paramètres de simulation (forme d'onde et SNR).

Pour se faire nous allons réaliser différentes simulations en utilisant notamment des simulations des étapes précédentes.

```
String[] argsBis = {"-s","-mess","1000","-form","NRZT","-ampl","-2","2","-snr","1","-ti","10","2","-codeur"};
```

Voici la simulation que nous avons réalisé, cette simulation reprend les paramètres que nous avons utilisé lors de la simulation des tests avec le multi-trajets. De cette manière nous pourrons constater ou non l'amélioration du TEB avec cette nouvelle fonctionnalité.

NRZT10	NRZTcod	NRZT
0,204	0,108	0,161
0,177	0,101	0,0865
0,151	0,07	0,06
0,142	0,059	0,04
0,123	0,0379	0,0315
0,096	0,023	0,019
0,07	0,014	0,01
0,046	0,0049	0,0045
0,037	0,001	0,008
0,019	0	0,001
0,014	0	0

Illustration 37: NRZT - Valeurs TEB avec decodeur

A l'aide de ces valeurs nous avons tracé trois courbes :

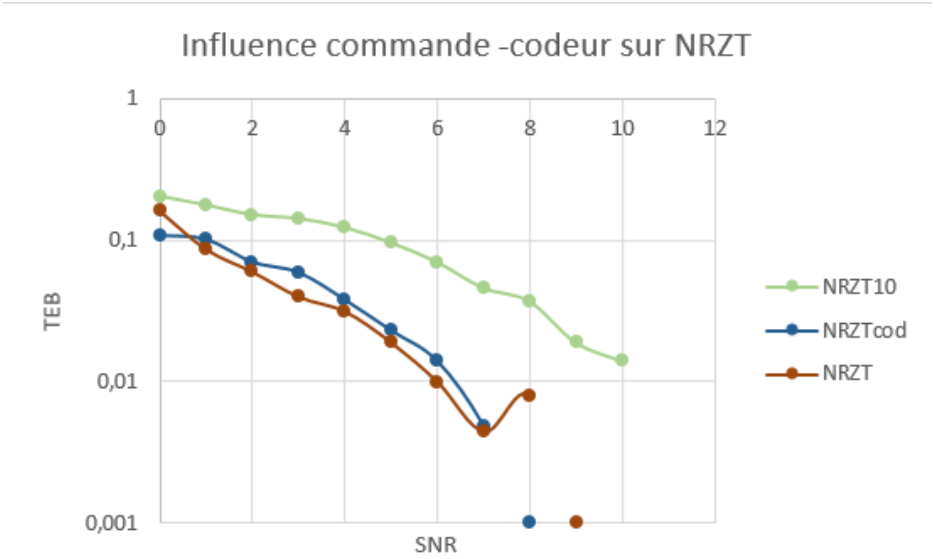

Illustration 38: NRZT - Influence commande -codeur

En bordeaux nous avons affiché la courbe d'un codage NRZT en le faisant varier de 0 à 10 de SNR sans ajouter de multi-trajets et de codeur. En vert nous avons la courbe obtenue à l'étape 4 qui affiche le signal NRZT avec un décalage temporel de 10 échantillons. Enfin en bleu nous avons la courbe du signal NRZT après avoir utilisé la nouvelle fonctionnalité codeur.

Ces trois courbes nous permettent de valider l'objectif de cette étape, en effet nous voyons bien que la courbe d'évolution du TEB NRZTcod se rapproche et suit la courbe NRZT, qui est une courbe n'ayant pas subi de décalage temporel. Ainsi nous en arrivons à la conclusion qu'utiliser le codeur permet d'améliorer le TEB et de réduire les erreurs introduites par le phénomène de multi-trajets.

IX. OBSERVATIONS

Cette partie nous a permis d'intégrer le codage de canal dans notre chaîne de transmission à l'aide d'une nouvelle commande unique. Nous avons pu au travers de simulation mesurer l'intérêt de ce codage sur le TEB, en observant notamment qu'il réduisait les effets du multi-trajets en améliorant même l'ensemble de la chaîne de transmission. En effet nous observons qu'il réduit aussi les effets du bruit gaussien ajouté à notre système.

X. ANNEXE

X.1 Diagramme de Classes étape 2

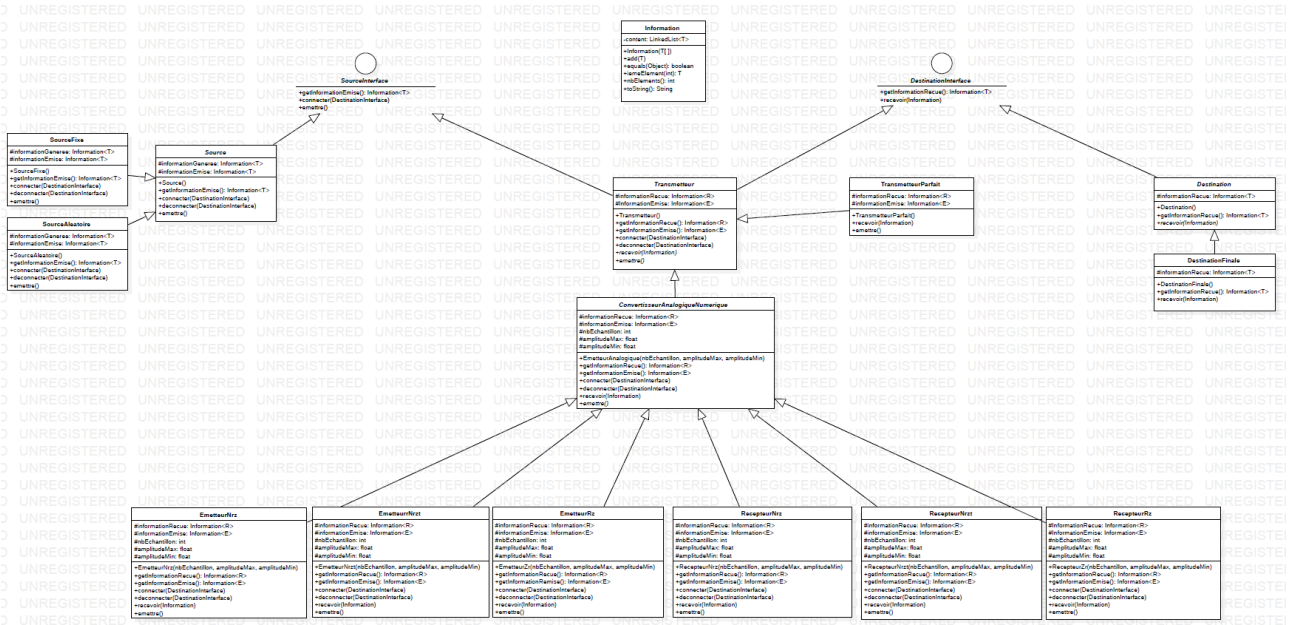

Illustration 39: Diagramme de classes étape 2

X.2 Diagramme de classes étape 3

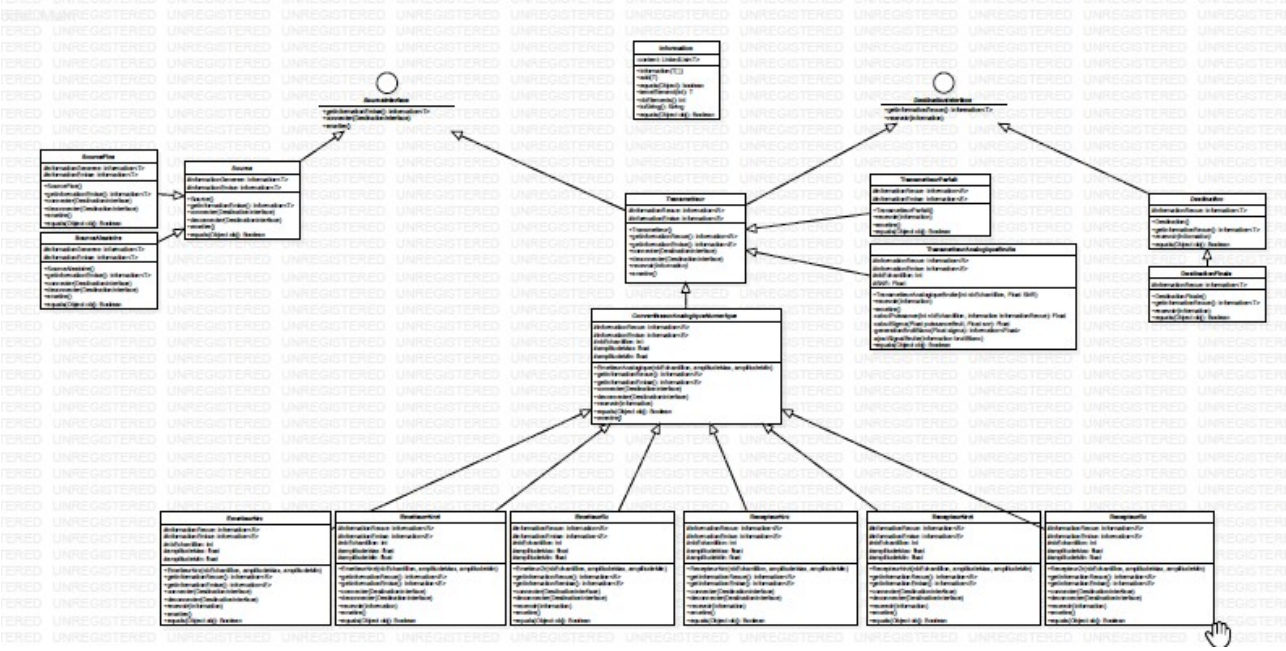

Illustration 40: Diagramme de classes étape 3

X.3 Diagramme de Classes étape 4a – Ajouts

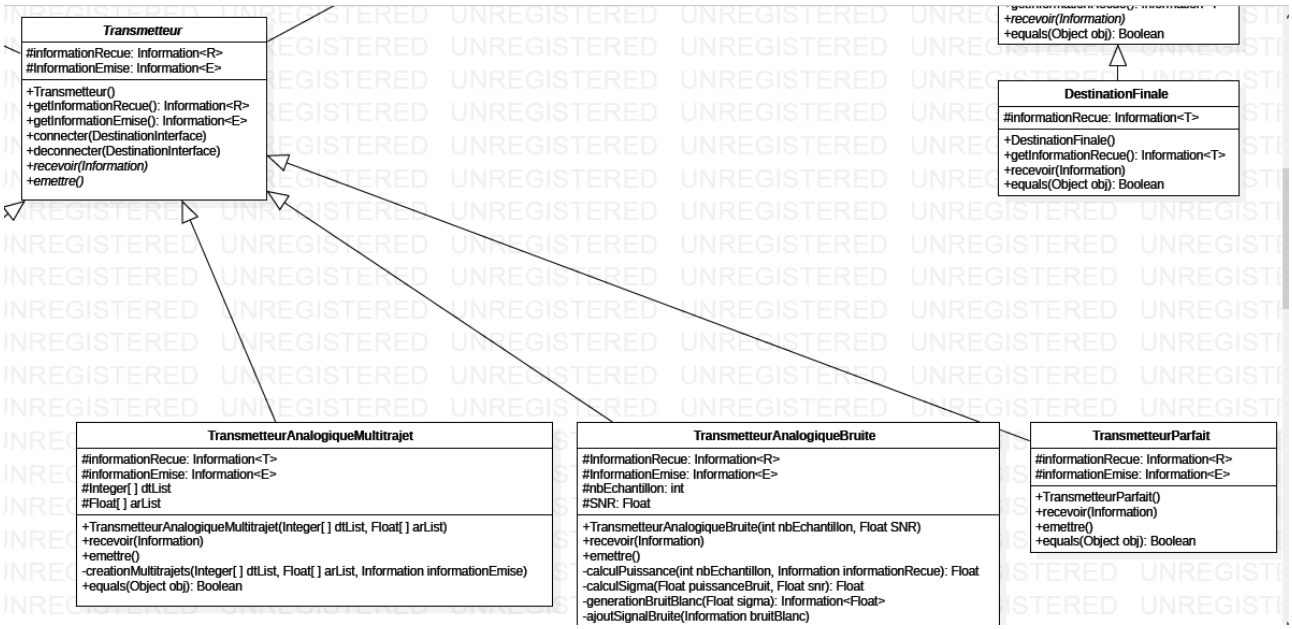

Illustration 41: Diagramme de classes étape 4a - TransmetteurAnalogiqueMultitrajet

X.4 Diagramme de classes – étape 5