Décodage d'images numériques issues de satellites météorologiques en orbite basse : le protocole LRPT de Meteor-M2 (partie 3/3)

J.-M Friedt, H. Boeglen, 21 mai 2019

Nous étudions un dernier élément de la chaîne de décodage des images de Meteor-M2 par la correction d'erreurs par bloc qu'est Reed Solomon. Son exploration sera l'opportunité d'approfondir quelques concepts de communication numérique, et en particulier justifier l'utilisation conjointe des codes convolutifs et des codes de Reed-Solomon compte tenu du bilan de liaison entre le satellite et le sol et d'un objectif de taux d'erreur de réception borné.

Nous avons atteint dans l'article précédent de cette série notre objectif, décoder une image issue de Meteor-M2 en partant du signal radiofréquence brut, en le transposant en bande de base puis en traitant le flux de données complexes résultant jusqu'à atteindre une image en passant par les diverses couches OSI pour aborder les codes convolutifs, les paquets de données et les imagettes JPEG, mais en cachant un dernier é'ément de la chaîne de décodage : la correction d'erreurs par code en bloc. Nous avions conclu notre précédent article [1] par la référence à l'article fondateur de Shannon sur la théorie de l'information, aussi cité en introduction de [2], en mentionnant qu'un code suffisamment complexe permet d'atteindre la limite théorique de capacité de communication C d'un canal bruité de rapport signal à bruit SNR et de bande passante W de $C = W\log_2(1+SNR)$. Le problème de l'implémentation d'une liaison de communication numérique est d'atteindre cette borne : l'article de Shannon dit simplement qu'un code "suffisamment complexe" permet d'atteindre ce but, sans l'expliciter plus que de dire qu'il faudra diluer l'information dans le temps pour permettre de corriger des erreurs ponctuelles qui s'insèreraient dans les bits transmis. Nous nous proposons ici d'aborder le codage d'information par blocs en vue de la correction d'erreurs de transmission [3], et à cette occasion de justifier des modes de modulation "simples" utilisés dans les liaisons spatiales. Nous appuierons nos affirmations sur le cours proposé à [4] et l'article du même auteur [5]. Les codes en blocs sont partout autour de nous – des compact discs (CD) aux mémoires flash [6] en passant par les codes QR [7] qui ont largement été décrits dans ces pages.

1 Débits de communication théoriques

Une analyse classique de l'équation de Shannon porte sur les deux régimes asymptotiques de propagation de communication qui vont justifier du mode de modulation utilisé en communication spatiales telles que nous les avons déjà vus – modulation de phase binaire (BPSK) pour GPS et modulation de phase à quatre état (QPSK) ici. Le rapport signal à bruit dans l'équation de Shannon est le ratio de la puissance transmise P_E par l'émetteur chargé de communiquer sur le bruit du canal. Prenons le cas simple du bruit thermique uniforme sur la bande de fréquence considérée : le bruit thermique est donné, comme dans tout problème de thermodynamique, par le constante de Boltzmann k_B multiplié par la température (en Kelvin). Ce bruit par unité spectrale (W/Hz) est multiplié par la bande passante du canal W pour donner la puissance du bruit $N = k_BT \times B$ et $SNR = \frac{P_E}{k_BTW}$. Dans ces conditions, l'équation de Shannon devient $C = W\log_2\left(1+\frac{P_E}{k_BTW}\right)$.

Les deux régimes de communication explicités dans [4] sont ceux à fort rapport signal à bruit (par exemple CPL ou DSL) limités par la bande passante, et à faible rapport signal à bruit limité par la puissance, cas qui nous intéressera pour une transmission de télévision numérique terrestre mais surtout ici dans une liaison spatiale. Dans le premier cas, SNR est grand donc $1+SNR \simeq SNR$ et doubler la bande passante $W' = 2W$ se traduit par $C' = W'\log_2(1+\frac{P_E}{k_BTW'}) = 2W\log_2\left(1+\frac{P_E}{2k_BTW}\right) \simeq 2W\log_2\left(\frac{P_E}{2k_BTW}\right) = 2W\left(\log_2\left(\frac{P_E}{k_BTW}\right)-1\right)$ et puisque SNR est grand, nous supposons $\log_2\left(\frac{P_E}{k_BTW}\right) \gg 1$ donc nous négligeons le terme -1 et il reste $C' = 2W\log_2\left(\frac{P_E}{k_BTW}\right) = 2C$ donc doubler la bande passante double la capacité de communication. Ainsi, plus les modes de modulation seront complexes en occupant toute la bande spectrale fournie par la norme allouée à un mode de communication donné, plus le débit sera élevé : c'est ainsi que WiFi adapte sa constellation (le nombre d'états transmis à chaque intervalle de communication) au rapport signal à bruit afin de maximiser le débit en fonction de la qualité de la liaison radiofréquence.

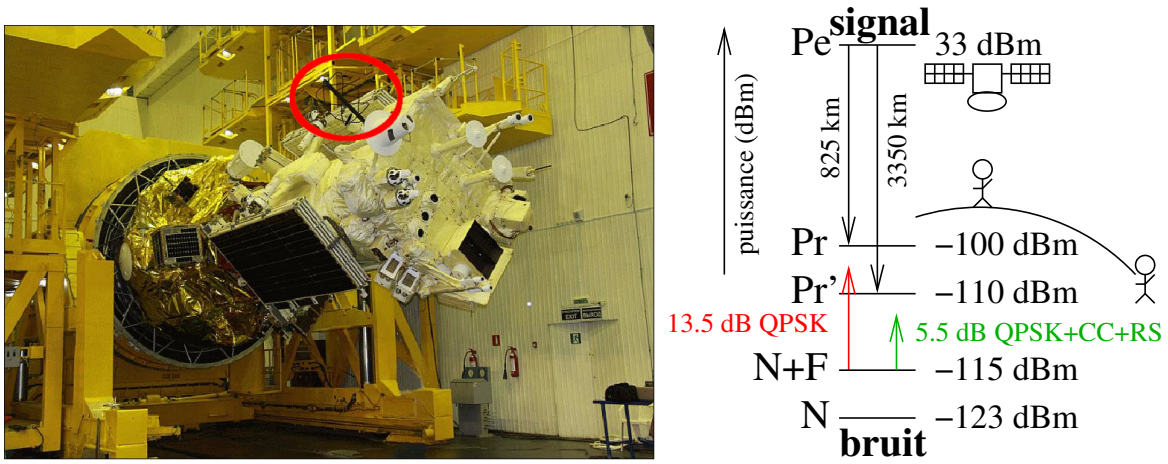


FIGURE 1 – Gauche : photographie de Meteor-M2 présentant son antenne omnidirectionnelle VHF émettant en polarisation circulaire, prise sur `directory.eoportal.org/web/eoportal/satellite-missions/m/meteor-m-2`. Droite : bilan de liaison radiofréquence reliant la puissance émise P_e à la puissance reçue P_r ou P'_r d'une part, et d'autre part le bruit thermique N complété de la contribution du récepteur F, avec le rapport signal à bruit nécessaire pour atteindre moins d'un bit erroné pour un million de bits transmis lorsque le satellite est à l'azimuth (rouge, distance de 825 km du récepteur) et à l'horizon (vert, distance de 3350 km du récepteur), justifiant des codes correcteurs d'erreur. L'écart entre le rapport signal à bruit fourni sur ce graphique (13,5 et 5,5 dB) et l'axe des abscisses de la Fig. 7 vient de l'efficacité spectrale qui relie ces deux grandeurs, soit $10\log_{10}(2) = 3$ dB pour QPSK.

Au contraire, si SNR est petit, alors nous considérons le développement limité du logarithme autour de 1 qui dit que $\log(1+x) \simeq x$ si $x \simeq 0$ et dans ce cas $C \simeq W \times \frac{P_E}{k_B T W} = \frac{P_E}{k_B T}$. La bande passante de communication a disparu de l'équation et seule la puissance émise importe pour déterminer la capacité de communication. C'est ce cas qui nous intéresse dans les liaisons spatiales.

Le mode de modulation est sélectionné de façon rationnelle en considérant le débit de communication et le rapport signal à bruit. Le rapport signal à bruit est défini d'une part par le niveau de signal reçu au sol, déterminé comme la puissance émise par le satellite, distribuée sur une sphère centrée sur le satellite (conservation de l'énergie) avant d'atteindre la surface de la Terre (équation de Friis). D'autre part, le niveau de bruit est déterminé par l'agitation thermique dans les composants électroniques, relevé d'un facteur de bruit représentatif des diverses contributions additionnelles dans un récepteur. Finalement, connaissant le rapport signal à bruit et la volume de données à transmettre sans erreur sur un bit – la taille d'une image compressée JPEG transmise par le satellite – nous en déduirons le taux d'erreur binaire et donc le mode de modulation approprié.

Concrètement

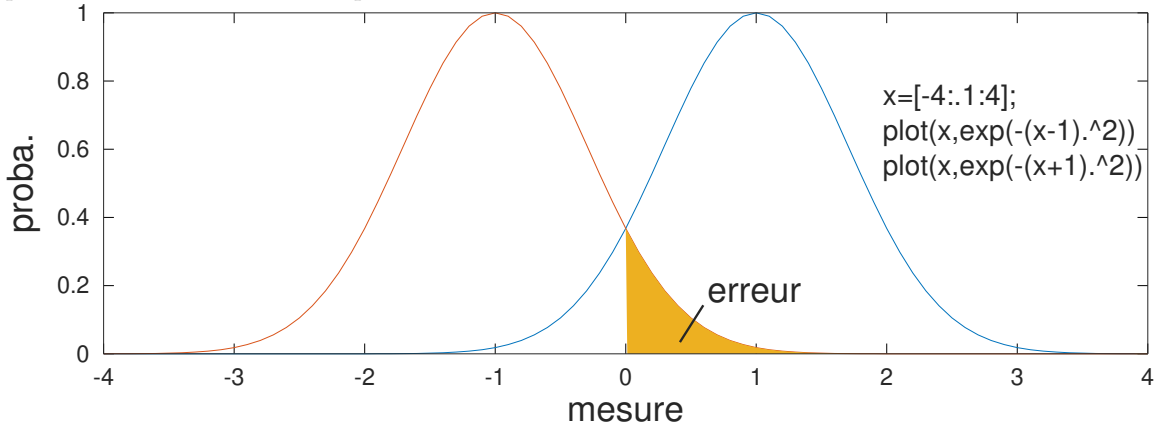
1. la page Wikipedia de LRPT nous informe que ce protocole indique une puissance émise de l'ordre de 2 W, ou $10log_{10}(2000) = 33$ dBm.
2. Meteor-M2 orbite la Terre à une altitude d de 825 km donc la puissance se distribue sur une sphère centrée sur le satellite de rayon 825 km avant d'atteindre au sol une antenne de dimension typique de la longueur d'onde $\lambda = c/f$ avec $f = 137,9$ MHz la fréquence du signal émis et $c = 3 \cdot 10^8$ m/s la célérité de la lumière. En normalisant par 4π stéradians la surface entourant les antennes d'émission et de réception, la puissance reçue P_r est une fraction de la puissance émise P_e déterminée par l'équation de Friis

$$\frac{P_r}{P_e} = \frac{\lambda^2}{(4\pi)^2 d^2} \Rightarrow \log_{10}\left(\frac{P_r}{P_e}\right) = 20\log_{10}(f) + 20\log_{10}(d) - 147,6$$

en exprimant f en Hz, d en m et en considérant que $\lambda = c/f$ d'où $10\log_{10}\left((c/4\pi)^2\right) = 147,6$. En l'absence de gain d'antenne (Fig. 1), la puissance reçue au sol est donc de $33 - 133.5$ dB$\simeq -100$ dBm (ou 0,1 pW !)

3. le signal est émis au rythme de $f_b = 72$ kb/s dans une bande passante d'environ $B = 120$ kHz. Ainsi, le bruit thermique est intégré sur cette bande pour déterminer un niveau de bruit $N = k_B \cdot T \cdot B$ avec $k_B = 1,38 \cdot 10^{-23}$ J/K la constante de Boltzmann, et $T = 298$ K la température absolue au sol. Plus facile à retenir, $10 \log_{10}(k_B T) = -204$ dBW$= -174$ dBm (passage des Watts aux milliwatts) donc la puissance du bruit est $-174+10 \log_{10}(125 \cdot 10^3) = -174+51 = -123$ dBm.
4. Par conséquent, le rapport signal à bruit est $-100 + 123 = 23$ dB.
5. Notre objectif est de transmettre une image de taille typique de 125 kB=1 Mb sans erreur, donc un taux d'erreur binaire inférieur à 10^{-6}. Admettant que rapport E_b/N_0 est lié au rapport signal à bruit par l'efficacité spectrale définie comme le rapport du débit de données f_b à la bande spectrale B qui vaut 2 pour la modulation QPSK, et tenant compte d'une dégradation du rapport signal à bruit de l'ordre de 8 dB par le récepteur radiofréquence, nous constatons sur la Fig. 2 que **la modulation QPSK seule suffit**, avec $E_b/N_0 = 10,5$ dB, à respecter le taux d'erreurs de 10^{-6} puisque $E_b/N_0 = 10,5 \Rightarrow SNR = 13,5$ dB compte tenu de l'efficacité spectrale de QPSK et $23 - 8 = 15$ dB en tenant compte du facteur de bruit du récepteur. **Une modulation plus complexe M-PSK avec $M > 4$ augmente le rapport f_b/B et interdira de respecter la taux d'erreur visé** (Fig. 2).
6. Le passage au zénith du satellite, lorsqu'il est à 825 km du récepteur, est le cas le plus favorable. À l'horizon, le bilan de liaison est dégradé car le satellite est plus loin de l'observateur, sans compter l'absorption de l'atmosphère. Lorsque le satellite passe sur l'horizon, sa distance r à l'observateur vérifie $r^2 + R^2 = (R + d)^2$ avec R le rayon terrestre, soit une distance $r = \sqrt{(R + d)^2 - R^2} = 3350$ km. Dans ces conditions, les pertes de propagation s'élèvent à 145 dB ou 12 dB de plus qu'à l'azimuth, et le rapport signal à bruit de 11 dB est **devenu insuffisant pour respecter la contrainte du taux d'erreur** de 10^{-6} compte tenu du bruit du récepteur (8 dB) et de l'efficacité spectrale (3 dB).
7. en nous référant à Fig. 7, nous constatons que **l'utilisation des codes de convolution et des codes par blocs permet de gagner 8 dB et de compenser l'augmentation de distance** entre le satellite et le récepteur entre le zénith et l'horizon. La justification de la complexité des encodages est donc ainsi démontrée.

Le point clé d'une liaison numérique radiofréquence tient en la détection du bit transmis, ou plutôt l'erreur de détection de celui-ci. Dans un canal de communication parfait, un bit x est émis et la réception y est identique à x, il n'y a aucune erreur possible. Dans un canal de communication pollué par du bruit uniformément distribué (supposé gaussien – en opposition aux pertes ponctuelles de liaison par interférence destructive des ondes radiofréquences se réfléchissant sur des obstacles par exemple), la réception $y = x + n$ est affectée d'un bruit n. La question de la détection se porte alors sur la décision permettant d'associer y au x le plus probable, ce qui se traduit en termes géométriques de trouver le point de la constellation sur laquelle se distribuent les x le plus proche de y [8]. Pour une modulation à deux états par exemple (BPSK – utilisé par GPS par exemple), ce problème s'illustre facilement par



Sur ce graphique, les deux états possibles des bits transmis sont +1 et -1, et un bruit gaussien de

variance unitaire affecte la transmission. La décision d'affecter y à +1 ou -1 se divise en 0, et l'erreur de transmission se produit si, lors de la transmission d'un -1, la valeur mesurée est positive (zone orange). Cette probabilité d'erreur est la queue de la gaussienne rouge au-dessus de 0. L'intégrale de la queue d'une gaussienne est une fonction tellement classique dans les problèmes de statistique qu'elle a un nom : c'est la fonction $Q(x)$ définie comme $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-u^2/2)du$ (i.e. l'intégrale de la gaussienne entre x et l'infini), tabulée ou implémentée dans la boîte à outils `communications` de GNU/Octave sous forme de `qfunc()` ou bien par la fonction d'erreur complémentaire `erfc()` puisque $Q(x) = \frac{1}{2} erfc(x/\sqrt{2})$. Fort de cette connaissance, nous pouvons tracer (Fig. 2) le taux d'erreur de communication connaissant la puissance du bruit et la distance entre les points de la constellation portant les x, tel que décrit par exemple en détail dans [9] : le tracé du taux d'erreur binaire de communication (TEB, ou BER en anglais) en fonction du rapport de l'énergie contenue dans chaque symbole transmis sur la puissance du bruit (E_b/N_0) s'obtient par des courbes de la forme $BER \simeq A \cdot Q(\sqrt{B \cdot E_b/N_0})$ avec les diverses modulations et codages affectant le facteur de proportionnalité A devant la fonction Q et son argument B. Nous retrouvons bien sur ce graphique le résultat présenté en [9, Fig.4] ainsi que [2, Fig. 6].

Ces expressions théoriques du taux d'erreur ne sont pas toujours disponibles et une simulation sur un grand nombre de bits transmis et aléatoirement corrompus (méthode de Monte-Carlo) par le canal de transmission est alors nécessaire pour établir le taux d'erreur. Une telle fonctionnalité est par exemple fournie par la bibliothèque IT++ disponible à `itpp.sourceforge.net` ou comme paquets Debian `libitpp8v5` et `libitpp-dev`. Le taux d'erreur d'une liaison BPSK est décrit comme exemple d'application à `itpp.sourceforge.net/4.3.1/bpsk.html` et le gain amené par le code de convolution à `itpp.sourceforge.net/4.3.1/convcode.html`. Cet outil est utilisé pour générer les courbes présentées sur Fig. 7.

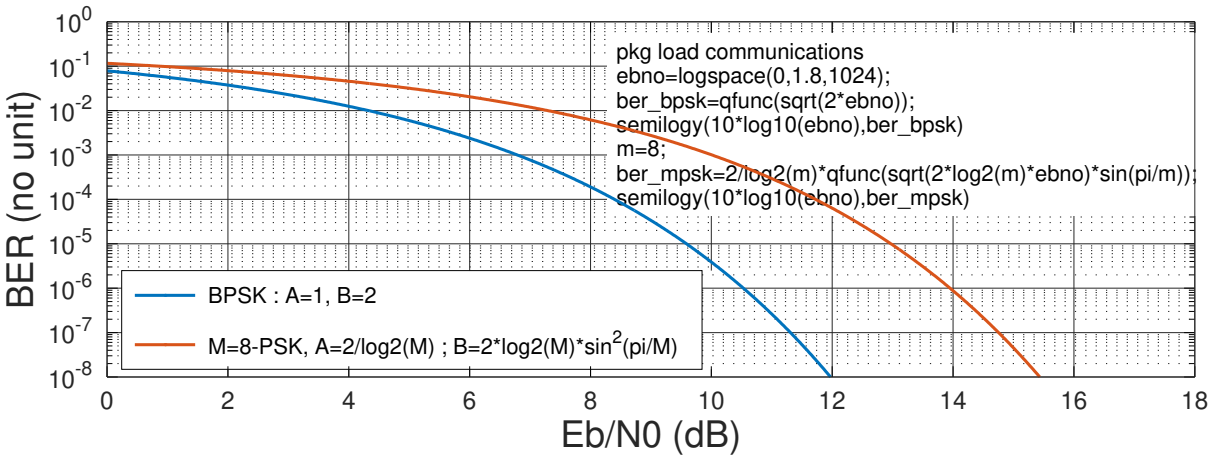


FIGURE 2 – Taux d'erreur binaire (TEB, ou *BER* pour son acronyme anglais) en fonction du rapport signal à bruit ou ici du rapport de l'énergie par bit au bruit. BPSK et QPSK présentent pratiquement les mêmes comportements.

2 Coder pour s'affranchir de l'impact du bruit

Les concepts de codage, convolutif et par bloc, que nous essaierons d'introduire dans la suite de cette présentation visent à identifier un codage dans lequel la distance entre les divers symboles transmis se maximise. De cette façon, le récepteur du signal entaché de bruit maximise ses chances d'identifier le message transmis malgré la corruption par le canal de communication. Alors que cette distance entre les symboles peut se définir de diverses façons, nous pouvons intuiter le concept en proposant une illustration sur la distance euclidienne familière à chacun (distance entre deux points dans l'espace). Si nous considérons une modulation de phase par exemple, nous pouvons transmettre diverses séquences de bits sur divers niveaux de phase imprimés sur une porteuse. Dans le plan complexe, un phase φ se présente comme un point de distance unitaire à l'origine (cercle unitaire dans le plan complexe) et d'angle φ par rapport à l'origine : le vecteur $I + jQ$ s'écrit $1 \times \exp(j\varphi)$. Si les phases sont uniformément distribuées sur N valeurs, alors deux points adjacents dans le plan complexe seront séparés (Fig. 3) d'une distance $d = \sin(\pi/N)$ (il suffit pour s'en convaincre de regarder les points autour de $1 + j0$). Chaque symbole reçu permet de récupérer $\log_2(N)$ bits (par exemple deux bits en QPSK). Plus N sera grand, plus le nombre de bits transmis à chaque période du signal numérique est grand, mais plus les points de la constellation seront rapprochés et le risque de se tromper crît. L'efficacité spectrale est le ratio du débit de communication (rythme binaire en bit/s) sur la bande passante occupée (Hz).

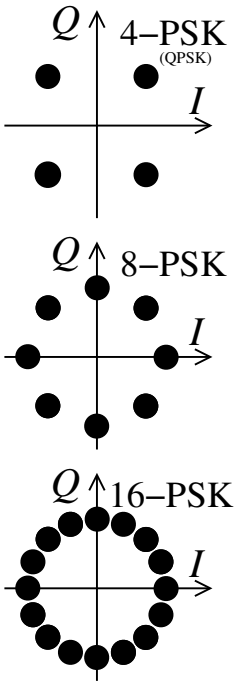


Figure 3: Augmenter le nombre de bits transmis à chaque période du signal numérique augmente l'efficacité spectrale mais augmente aussi le risque de se tromper lors de l'assignation du signal reçu bruité (extension du cercle noir) à une valeur possible de la constellation.

Si maintenant au lieu de transmettre $\log_2(N)$ par symbole nous encodons le message à transmettre pour que plusieurs (P) bits successifs représentent le message transmis. Alors nous pouvons considérer le problème dans un espace de dimension P : un façon d'intuiter le gain de ce codage est de considérer maintenant la distance entre les points de la constellation dans ce nouvel espace. Par exemple pour $P = 3$, nous nous retrouvons dans la dimension familière de l'espace à 3 dimensions où les 2^3 états possibles se distribuent sur un cube. En ne sélectionnant qu'un sous-ensemble "judicieux" des sommets de ce cube, nous pouvons augmenter la distance minimale entre ces points et réduire le risque d'assigner le signal reçu bruité au mauvais symbole émis. Par exemple pour le cube, ne prendre que les sommets diamètralement opposés sur chaque face augmente la distance minimale entre symbole de $\sqrt{2}$ la longueur de la diagonale du carré formant chaque face du cube.

Le même problème se retrouve pour le stockage de masse de mémoires non-volatiles : le passage de SLC (*Single Level Cell*) à MLC (*Multi-Level Cell*) dans les technologies de stockage par portes NAND [10] augmente la densité d'états stockés et donc la capacité des disques, mais au détriment de la robustesse et de la sensibilité aux erreurs : une migration de charges sur les électrodes mémorisant l'état se traduit par un changement de potentiel et donc une corruption d'un état qu'il faut pouvoir corriger. Ici encore les codes correcteurs d'erreur par bloc sont massivement exploités pour éviter la corruption des données stockées et re-écrire des données valides si une erreur est détectée à la lecture.

3 Code correcteur d'erreurs par bloc de Reed Solomon

Le codage par convolution a pour vocation de compenser du bruit distribué sur des bits du fait de la liaison radiofréquence bruitée entre l'émetteur et le récepteur, et fait l'hypothèse d'un bruit uniforme aléatoire qui peut impacter chaque bit indépendamment de ses voisins. Il ne saurait cependant corriger des blocs de données corrompues par un interférent ponctuel : ce type d'erreur est pris en charge par la correction par bloc, par exemple de Reed-Solomon. Ce code s'apparente à la correction par blocs que nous avions déjà vu dans RDS avec le codage BCH [11]. Ici, chaque paquet de 255 octets est formé de 223 octets utiles et 32 octets de code correcteur, qui permettent d'identifier une erreur de transmission et d'en corriger une partie. Ce type de code correcteur s'appelle donc RS(255,223) puisque sur 255 octets transmis, 223 sont des données et donc les 32 derniers sont le code correcteur d'erreur. `libfec` fournit la bibliothèque nécessaire à la correction d'erreurs par RS(255,223), tel que décrit dans

[12]. Comme le but de la correction par bloc est de corriger une série de symboles erronés, il est judicieux de distribuer l'information le long de la trame transmise afin de minimiser l'impact de l'interférence qui affecte plusieurs bits adjacents. Ainsi, au lieu de diviser la trame de 1020 octets de long en 4 trames adjacentes de 255 octets, le choix est fait d'entrelacer les 4 séquences de données et leur code correcteur d'erreur tel que illustré en Fig. 4, avec le code correcteur d'erreur des 4 séquences placé en fin de trame. Nous allons nous focaliser ici sur le décodage et la correction des symboles erronés, l'encodage état très pédagogiquement décrit dans [13] dans le contexte de la transmission d'images de télévision numérique terrestre.

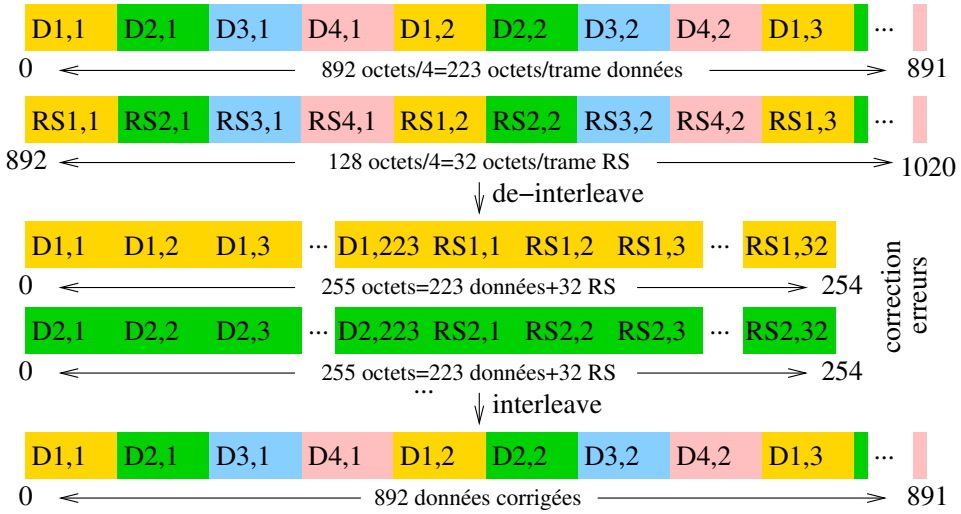


FIGURE 4 – Organisation des données le long d'une trame CVCDU de 1020 octets de long (nous avons déjà éliminé le mot de synchronisation de 4 octets en entête). Les 892 premiers octets contiennent les données `D` qui seront considérées comme entrelacées pour les 4 séquences de 223 octets corrigeables par Reed Solomon RS(255,232), et les derniers 128 octets contiennent, toujours de façon entrelacée, les 4 séquences de 32 octets de codes de correction `RS`. Appliquer la correction nécessite donc de dé-entrelacer les données (haut → milieu), appliquer Reed Solomon pour identifier et corriger les erreurs (milieu), et re-enterlacer les données (milieu→bas) pour les replacer dans leur ordre d'origine, mais après correction des octets potentiellement corrompus lors de la liaison radiofréquence.

Nous pouvons nous entraîner dans un premier temps pour comprendre l'implémentation de Reed Solomon dans `libfec` :

```
1  #include <fec.h>    // gcc -o jmf_rs jmf_rs.c -I./libfec ./libfec/libfec.a
2
3  int main()
4  {int j;
5   uint8_t rsBuffer[255];
6
7   uint8_t tmppar[32];
8   uint8_t tmpdat[223];
9
10  for (j=0;j<255; j++) rsBuffer[j]=(rand()&0xff);  // received data
11  for (j=0;j<223;j++)  tmpdat[j]=rsBuffer[j];      // backup data
12  encode_rs_ccsds(tmpdat,tmppar,0);                // create RS code
13  for (j=223;j<255;j++) rsBuffer[j]=tmppar[j-223]; // append RS after data
14  rsBuffer[42]=42; tmpdat[42]=42;                  // introduce errors
15  rsBuffer[43]=42; tmpdat[43]=42;                  // ... on purpose
16  rsBuffer[44]=42; tmpdat[44]=42;    // ... to check correction capability
17  rsBuffer[240]=42;tmppar[240-223]=42;
18  printf("RS:%d\n",decode_rs_ccsds(rsBuffer, NULL, 0, 0)); // check that RS can correct
19  for (j=0;j<223;j++)
```

```
20    if (rsBuffer[j]!=tmpdat[j]) {printf("%d:␣%hhd␣->␣%hhd␣;␣",j,tmpdat[j],rsBuffer[j]);}
21  for (j=223;j<255;j++)
22    if (rsBuffer[j]!=tmppar[j-223]) {printf("%d:␣%hhd␣->␣%hhd␣;␣",j,tmppar[j-223],rsBuffer[j]);}
23  }
```

dans cet exemple nous créons des données (aléatoires) que nous encodons, puis nous modifions 4 données de la charge utile et 1 donnée du code correcteur, et testons la capacité de correction du décodeur. Le résultat,

```
RS:4
42: 42 -> 5 ; 43: 42 -> 23 ; 44: 42 -> 88 ; 240: 42 -> 95
```

est conforme à nos attentes : 4 erreurs sont identifiées et corrigées.

L'application du programme d'exemple sur les 128 octets de fin de trame chargés de corriger les 892 octets du début de trame ... ne fonctionne pas du tout ! Encore une astuce indiquée par Lucas Teske que nous n'avons pas trouvé dans les documentations : il s'agit d'un code correcteur *dual basis Reed Solomon* dans lequel les octets sont encore une fois passés dans une table de transposition tel que décrit dans `github.com/opensatelliteproject/libsathelper/blob/master/src/reedsolomon.cpp`. L'implémentation du code correcteur d'erreur pas blocs dans cette nouvelle base est justifiée dans [14, Tab.4] comme un gain significatif (66%) de ressources par rapport à une implémentation sur les données brutes. Une fois cette transposition effectuée, le code correcteur fonctionne, selon l'exemple ci-dessous qui inclut toute la séquence de décodage, à savoir algorithme de Viterbi, application (XOR) du polynôme pour retirer la distribution aléatoire des données, regroupement des paquets de données avec leur code correcteur de Reed Solomon, application du polynôme de transposition, correction d'erreur, retrait du polynôme de transposition pour finalement obtenir les données corrigées :

```
1  #include <fec.h> // gcc -o demo_rs demo_rs.c -I./libfec ./libfec/libfec.a
2
3  // github.com/opensatelliteproject/libsathelper/blob/master/src/reedsolomon.cpp
4  // dual basis Reed Solomon !
5  #include "dual_basis.h"
6
7  unsigned char pn[255] ={           // randomization polynomial
8      0xff, 0x48, 0x0e, 0xc0, 0x9a, 0x0d, 0x70, 0xbc, \
9      0x8e, 0x2c, 0x93, 0xad, 0xa7, 0xb7, 0x46, 0xce, \
10 [...]
11     0x08, 0x78, 0xc4, 0x4a, 0x66, 0xf5, 0x58  };
12
13 #define MAXBYTES (1024)
14
15 #define VITPOLYA 0x4F
16 #define VITPOLYB 0x6D
17
18 #define RSBLOCKS 4
19
20 #define PARITY_OFFSET 892
21
22 void interleaveRS(uint8_t *idata, uint8_t *outbuff, uint8_t pos, uint8_t I) {
23   for (int i=0; i<223; i++) outbuff[i*I+pos]=idata[i];
24 }
25
26 int viterbiPolynomial[2] = {VITPOLYA, VITPOLYB};
27
28 int main(int argc,char *argv[]){
29  int res,i,j,framebits,fdi,fdo;
30  unsigned char data[MAXBYTES],symbols[8*2*(MAXBYTES+6)]; // *8 for bytes->bits & *2 Viterbi
31  void *vp;
32  int derrors[4] = { 0, 0, 0, 0 };
33  uint8_t rsBuffer[255],*tmp;
34  uint8_t rsCorData[1020];
35
```

```
36 fdi=open("./extrait.s",O_RDONLY);
37 fdo=open("./sortie.bin",O_WRONLY|O_CREAT,S_IRWXU|S_IRWXG|S_IRWXO);
38 read(fdi,symbols,4756+8);  // offset
39 framebits = MAXBYTES*8;
40
41 do {
42  res=read(fdi,symbols,framebits*2+50);  // 50 additional bytes to finish viterbi decoding
43  lseek(fdi,-50,SEEK_CUR);               // go back 50 bytes
44  for (i=1;i<2*framebits;i+=2) symbols[i]=-symbols[i]; // I/Q constellation rotation
45  set_viterbi27_polynomial(viterbiPolynomial);
46  vp=create_viterbi27(framebits);        // convolution -> Viterbi
47  init_viterbi27(vp,0);
48  update_viterbi27_blk(vp,symbols,framebits+6);
49  chainback_viterbi27(vp,data,framebits,0);
50  tmp=&data[4];                          // rm synchronization header
51  for (i=0;i<1020; i++) tmp[i]^=pn[i%255];  // XOR decode (dual basis)
52
53  for (i=0; i<RSBLOCKS; i++)
54      { for (j=0;j<255; j++) rsBuffer[j]=tmp[j*4+i];        // deinterleave
55        for (j=0;j<255; j++) rsBuffer[j]=ToDualBasis[rsBuffer[j]];
56        derrors[i] = decode_rs_ccsds(rsBuffer, NULL, 0, 0); // decode RS
57        for (j=0;j<255; j++) rsBuffer[j]=FromDualBasis[rsBuffer[j]];
58        interleaveRS(rsBuffer, rsCorData, i, RSBLOCKS); // interleave
59        printf(":%d",derrors[i]);
60      }
61  write(fdo,data,4);                      // header
62  write(fdo,rsCorData,MAXBYTES-4);  // corrected frame
63 } while (res==(2*framebits+50));
64 close(fdi);
65 close(fdo);
66 exit(0);
67 }
```

Ce code est donc l'apothéose de toute la séquence de conversion des phases issues des coefficients I/Q vers les bits prêts à être assemblés pour retrouver les trames puis les images JPEG, en tenant compte des deux codes correcteurs d'erreur, par convolution et par bloc. Le résultat de cette correction additionnelle est illustrée en Fig. 5 et démontre comment l'ajout du code de correction par bloc permet d'étendre la plage d'analyse des images JPEG alors que le satellite s'approche de l'horizon.

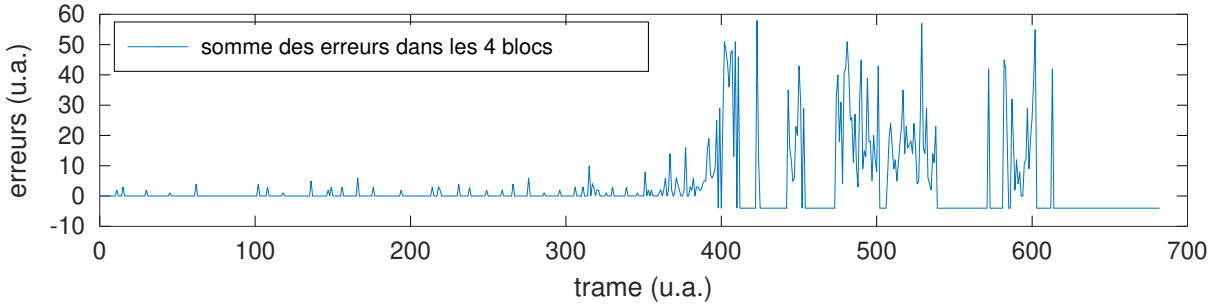


FIGURE 5 – Résultat de la correction d'erreurs au cours du passage du satellite. Nous constatons clairement que la correction par blocs devient d'autant plus efficace que le satellite s'abaisse sur l'horizon et que le bilan de liaison se dégrade. -1 indique qu'il y avait trop d'erreurs dans la réception pour permettre une correction des octets erronés.

4 Utilisation conjointe des codes convolutifs et des codes de Reed-Solomon

Nous avons présenté les caractéristiques des codes convolutifs dans la première partie de cette série d'articles consacrés à Meteor-M2. Les codes en blocs de Reed-Solomon (RS) sont quant à eux une extension des codes BCH pour les cas non binaires et sont très pratiques lorsque l'on travaille avec des données comme des octets (8 bits). La capacité de correction t de ce type de code était égale à $(n-k)/2$ avec n le nombre de symboles codés et k le nombre de symboles d'information. Ces codes sont très efficaces pour corriger des erreurs en rafales (*burst errors*) comme c'est le cas pour une rayure présente sur un CD ou un DVD ou lors d'une transmission sur un canal radiomobile. Il faut cependant préciser que le code RS travaille sur des mots de code définis dans GF(2^m) [15]. Ainsi, il sera incapable de corriger des erreurs binaires dispersées sur plus de t symboles même si ce nombre d'erreurs reste inférieur à $t \times 2^m$. À l'opposé, les codes convolutifs sont très efficaces pour corriger des erreurs dispersées. Dans le cas d'erreurs en rafales, ils se désynchronisent et le décodage échoue.

On comprend que dans ces conditions on peut être tenté par l'association de ces deux codes pour d'une part, pouvoir bénéficier des avantages de chacun d'eux et d'autre part, pour disposer d'un gain de codage plus important. Cette idée a été formalisée par Dave Forney lors de sa thèse au MIT en 1965 [16]. On retrouve depuis cette association code en blocs/code convolutif dans de nombreuses normes de télécommunications. Ce type d'association est appelé concaténation série et est représentée sur la Figure 6. Afin d'étendre les capacités de correction de l'un des codes on ajoute également un entrelaceur. Il existe d'autres méthodes de concaténation plus sophistiquées que nous n'aborderons pas ici et qui concernent les codes approchant la capacité de Shannon comme les Turbo codes. Même si les gains de codage obtenus lors de l'association ne s'additionnent pas directement cela conduit à un code plus performant car il produit des mots de codes très longs qui sont générés de manière "aléatoire" du fait de l'entrelaceur. Étonnamment, la distance minimale du code concaténé n'est pas significativement augmentée. Pourtant le nombre moyen de bits erronés correspondant à des évènements d'erreurs de distance faible est très bas. Ce nombre est essentiellement réduit proportionnellement à la profondeur de l'entrelaceur.

FIGURE 6 – Concaténation série

Mais au fait qu'est-ce que l'entrelacement ? Il s'agit simplement d'une opération de réorganisation périodique et réversible de blocs de L symboles transmis. Il existe plusieurs type d'entrelaceurs qui ont des caractéristiques particulières en fonction de l'utilisation prévue. Nous ne verrons ici que les entrelaceurs matriciels comme celui utilisé dans Meteor-M2. Pour faire simple, l'entrelacement consiste à rentrer les mots du code dans une matrice par ligne puis à lire la matrice par colonne. Un entrelaceur matriciel possède une profondeur J et une période L.

Définition 1 : (Profondeur). La profondeur J d'un entrelaceur est la séparation minimale exprimée en période de symboles à la sortie de l'entrelaceur entre deux symboles quelconques adjacents à l'entrée de l'entrelaceur.

Définition 2 : (Période). La période L d'un entrelaceur est la plus petite période de temps pour laquelle l'algorithme de réorganisation utilisé par l'entrelaceur se répète.

L'encadré donne un exemple d'entrelacement pour étendre la capacité de correction d'un code RS(255,223). C'est exactement ce principe qui est utilisé pour Meteor-M2 (cf. paragraphe 3).

La capacité de correction du code est de $t = (n–k)/2 = (255-223)/2 = 16$ mots. Nous souhaitons utiliser ce code sur un canal susceptible d'introduire des rafales d'erreurs de $d = 32$ mots. Il faut utiliser un entrelaceur de profondeur $L = ceil(d/t) + 1 = ceil(32/16) + 1 = 3$ mots du code.

Le programme GNU/Octave suivant

```
1 n=255;  % taille des mots du code
```

```
k=223;  % taille des donnees
b=32;   % nbre de symboles consecutifs susceptibles d'etre corrompus par le canal

p=n-k; % nbre de symboles de parite
t=p/2; % capacite de correction du code RS
D=ceil(b/t)+1; % profondeur de l'entrelaceur
memory = zeros(D,n); % init memoire de ljentrelaceur

data ='LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_';
data=char([repmat(data,[1,fix(n/length(data))]),data(1:mod(n,length(data)))]);
intlvrInput=repmat(data(1:n),[1 D]); % D blocs de donnees identiques transmis
fprintf('Donnees en entree de l''entrelaceur ->\n'); disp(char(intlvrInput));
for index=1:D  % ENTRELACEUR
    memory(index,1:end)=intlvrInput((index-1)*n+1:index*n);
end
intlvrOutput=zeros(1,D*n);
for index=1:n % lecture de la matrice colonne par colonne
    intlvrOutput((index-1)*D+1:index*D)=memory(:,index);
end
% cree b symboles d'erreur a partir de la position 25 : '*' signifie une erreur
intlvrOutput(1,25:24+b)=zeros(1,b)+42;
fprintf('\nSortie de l''entrelaceur apres corruption\n');
disp(char(intlvrOutput));

deintlvrOutput=zeros(1,D*n); % DESENTRELACEUR
for index=1:n % Ecriture dans le desentrelaceur colonne par colonne
    memory(:,index)=intlvrOutput((index-1)*D+1:index*D)';
end
for index=1:D % lecture du desentrelaceur ligne par ligne
    deintlvrOnput((index-1)*n+1:index*n)=memory(index,1:end);
end
fprintf('Sortie du desentrelaceur ->\n');
disp(char(deintlvrOnput));
```

illustre le fonctionnement de l'entrelaceur pour le cas que nous venons de traiter. On constate que l'entrelaceur disperse bien les rafales d'erreurs de sorte qu'elles ne dépassent pas 11 symboles pour un mot du code de 255 symboles. Le code RS est donc en mesure de corriger ces erreurs puisque RS(255,223) peut corriger 16 symboles.

Les données en entrée de l'entrelaceur sont

```
LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP
_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQU
ES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIMELINUX_
MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_
COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINU
X_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIMELINUX_MAGAZINE
_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICA
TIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZI
NE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME
```

qui après corruption par un burst d'erreurs de 32 symboles marquées par "*" deviennent

```
LLLIIINNNUUUXXX___MMMAAA********************************E___BBBEEEAAAUUUCCCOOOUUUPPP___L
LLEEESSS___CCCOOOMMMMMMUUUNNNIIICCCAAATTTIIIOOONNNSSS___NNNUUUMMMEEERRRIIIQQQUUUEEESSS__
_LLLIIINNNUUUXXX___MMMAAAGGGAAAZZZIIINNNEEE___AAAIIIMMMEEE___BBBEEEAAAUUUCCCOOOUUUPPP_
__LLLEEESSS___CCCOOOMMMMMMUUUNNNIIICCCAAATTTIIIOOONNNSSS___NNNUUUMMMEEERRRIIIQQQUUUEEESSS
___LLLIIINNNUUUXXX___MMMAAAGGGAAAZZZIIINNNEEE___AAAIIIMMMEEE___BBBEEEAAAUUUCCCOOOUUUPPP_
__LLLEEESSS___CCCOOOMMMMMMUUUNNNIIICCCAAATTTIIIOOONNNSSS___NNNUUUMMMEEERRRIIIQQQUUUEEESSS
___LLLIIINNNUUUXXX___MMMAAAGGGAAAZZZIIINNNEEE___AAAIIIMMMEEE___BBBEEEAAAUUUCCCOOOUUUPPP_
__LLLEEESSS___CCCOOOMMMMMMUUUNNNIIICCCAAATTTIIIOOONNNSSS___NNNUUUMMMEEERRRIIIQQQUUUEEESSS
___LLLIIINNNUUUXXX___MMMAAAGGGAAAZZZIIINNNEEE___AAAIIIMMMEEE
```

pour finalement, en sortie du désentrelaceur, apparaître comme

```
LINUX_MA**********_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP
_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQ
UES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIMELINU
X_MA**********_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LE
S_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_
LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIMELINUX_MA
**********E_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_CO
MMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX
_MAGAZINE_AIME_BEAUCOUP_LES_COMMUNICATIONS_NUMERIQUES_LINUX_MAGAZINE_AIME
```

Les 11 “*” pourront être corrigées par Reed-Solomon puisque $11 < 16$.

Maintenant que nous comprenons l’intérêt d’un entrelaceur, voyons si le placement des codes dans la concaténation série a une importance. Dans ce cas, on parle de code interne (*inner code*) pour le code le plus proche du canal et de code externe dans l’autre cas (*outer code*). Si les codes RS et convolutifs pouvaient être facilement décodés à l’aide d’un décodeur souple, l’ordre aurait moins d’importance. Cependant, ce n’est pas le cas pour les codes RS. Il existe bien des algorithmes de décodage souple mais ceux-ci sont soit sous-optimum (algorithmes de type Chase) soit complexes à mettre en œuvre (algorithme de Koetter-Vardy ; sous brevet). Par contre, l’algorithme de Viterbi pour le décodage des codes convolutifs est capable de traiter des données souples et est peu complexe. Rappelons, qu’il est important de pouvoir procéder à un décodage souple car le gain de ce dernier est de l’ordre de 2 dB en moyenne. Viterbi en 1991 soulignait [17] déjà l’importance de cette notion : “Never discard Information prematurely that may be useful in making a decision until after all decisions related to that information have been completed”. À titre d’illustration d’utilisation de ce concept lors de la perte de séquences de données et non pas de bits individuels comme l’induit un bruit gaussien, la NASA a utilisé pour ses liaisons avec les sondes en espace lointain (Deep Space Network) le réseau d’antennes Very Large Array (VLA) au Nouveau Mexique. Comme tout réseau d’antennes, des trames de maintenance, en particulier pour corriger la phase de l’oscillateur local de chaque antenne, sont périodiquement transmises entre les antennes, interrompant le flux de données de télémesure. Dans le cas du VLA, l’interruption est de 1.6 ms toutes les 50 ms : de telles pertes de données n’empêchent pas la réception des informations issues des sondes Voyager puisque les octets perdus sont rattrapables grace au code de Reed Solomon [18, p.352].

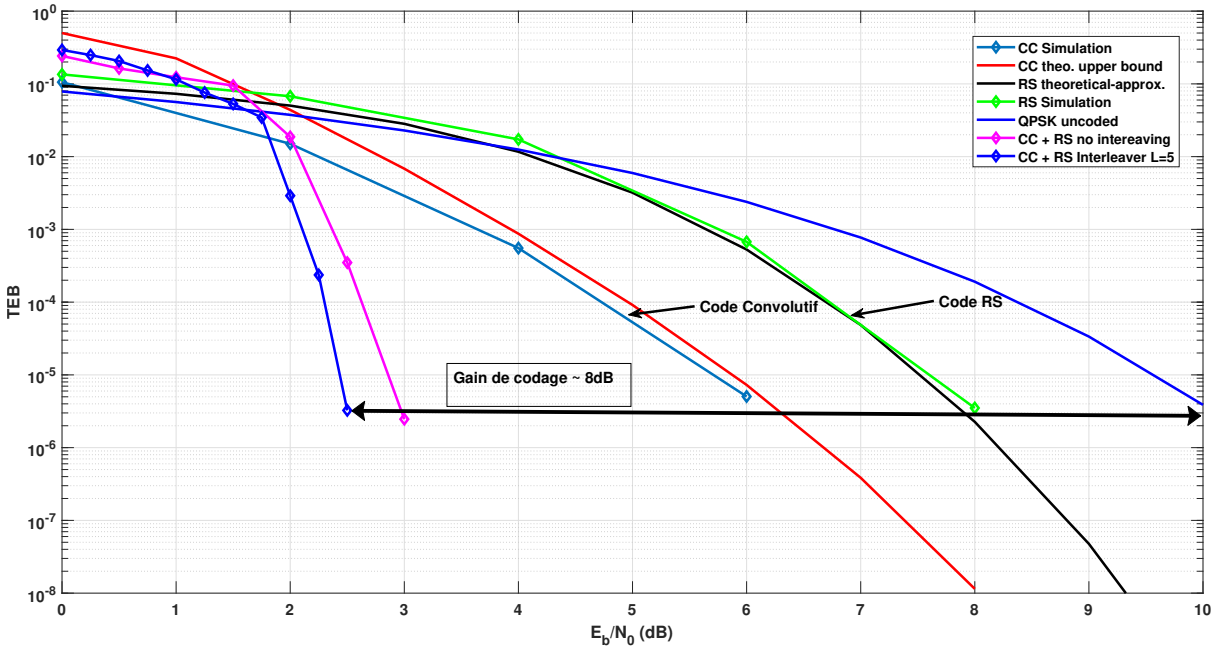


FIGURE 7 – Performances en terme de TEB du code concaténé CCSDS. Ces courbes sont générées par la bibliothèque IT++ et ses fonctions de simulation de communication numérique.

Le code convolutif (décodé par exemple par algorithme de Viterbi) sera donc le code interne de l'association. Afin d'étendre la capacité de correction du code RS on intercale un entrelaceur matriciel entre les deux codes. Ce dernier a une profondeur L variable qui est fonction de la taille du burst d'erreurs que l'on souhaite corriger. La Figure 7 présente les performances en terme de TEB du code concaténé standardisé par le CCSDS, à savoir un code RS(255,223) et un code convolutif (7,1,2) de matrice génératrice $G = [155\ 117]_8$ sans et avec entrelaceur avec $L = 5$. Nous présentons également les courbes de TEB des codes pris indépendamment. Lors du calcul du bilan de liaison Meteor-M2, nous avons constaté qu'il nous manquait environ 8 dB pour assurer une communication fiable lorsque le satellite se trouvait à 3350 km récepteur, au niveau de l'horizon. Nous remarquons sur les courbes de la Fig. 7 que pour un TEB de 10^{-6} le code RS procure un gain de 2 dB par rapport à la modulation non codée. Le code convolutif seul amène un gain de 4 dB. La concaténation de ces deux codes avec l'entrelacement permet d'obtenir 8 dB, exactement ce qui manquait à notre bilan de liaison ! On constate clairement le gain apporté par l'entrelaceur notamment pour les TEB faibles. En effet, ce dernier a pour effet d'augmenter la distance minimale des mots du code ce qui se traduit par un gain particulièrement important dans les zones à faible TEB. Vous aurez sans doute remarqué les faibles valeurs de $Eb/N0$ auxquelles travaille ce code. Pourtant, même si cette association est efficace pour Meteor-M2, ses performances sont encore à plus de 3 dB de la limite de Shannon ! Jusqu'en 1993, les ingénieurs en télécommunications considéraient que c'était une limite inatteignable et concevaient leurs systèmes de codage en prenant une marge de 3 dB par rapport à cette limite. Pourtant en 1993, grâce au concept de décodage souple itératif, appelé effet Turbo par leurs inventeurs, on s'approche enfin de cette limite. Le Turbo code est d'ailleurs un code concaténé parallèle d'un type un peu particulier. Nous aurons peut-être l'occasion de vous en dire un peu plus sur ce sujet passionnant dans un prochain article.

5 Conclusion

Cette exploration en trois parties de toutes les couches d'abstraction nécessaires au décodage d'images numériques issues de Meteor-M2 avait avant tout pour objectif de motiver l'étude des liaisons numériques. Une liaison spatiale est un cas idéal de bilan de liaison en espace libre, avec la motivation de savoir que le satellite est physiquement au dessus de la région observée (capteur *push-broom*). Sans prétendre présenter en détail la théorie sous-jacente de chaque partie, nous nous sommes focalisés sur une démonstration pratique qui peut fournir un contexte pour approfondir ses connaissances sur chaque point de l'acquisition du flux de données radiofréquences au décodage des bits puis des phrases pour enfin arriver aux images décompressées du format JPEG. Finalement la qualité de l'image est améliorée par la correction des blocs d'informations perdues lors de la transmission tel que nous venons de le voir dans ce dernier épisode.

On notera qu'un certain nombre de codes sources de cette série d'articles se sont vus tronquer, par soucis de compacité, des fichiers d'entête annonçant des bibliothèques standard et d'entrée sortie du C : le lecteur n'aura aucun mal à retrouver ces fichiers d'entête (`stdio.h`, `stdlib.h`, `unistd.h` ...) permettant d'ouvrir, lire, écrire ou fermer des fichiers ainsi que communiquer avec la console.

Références

[1] J.-M Friedt, *Décodage d'images numériques issues de satellites météorologiques en orbite basse : le protocole LRPT de Meteor-M2*, GNU/Linux Magazine France **226 & 227** (2019)

[2] H. Boeglen & L. Mura, *Les bases des communications numériques avec GNU Radio*, GNU/Linux Magazine **225** (2019)

[3] D.J. Costello, J. Hagenauer, H. Imai & S.B. Wicker, *Applications of Error-Control Coding*, IEEE Trans. on Information Theory **44**(6), 2531–2560 (1998) à `www.researchgate.net/publication/3079617_Applications_of_Error-Control_Coding`

[4] D. Forney & *al.*, *Principles of Digital Communication II* (2005) à `ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-451-principles-of-digital-communication-ii-spring-2005/`

[5] G.D. Forney & G. Ungerboeck, *Linear Gaussian Channels*, IEEE Trans. on Information Theory **44** (6), 2384– 2415 (1998) à `www-classes.usc.edu/engr/ee-s/568/org/Forney-Ung.pdf`

[6] Note technique *NAND Error Correction Codes Introduction*, Macronix (2017) à `www.macronix.com/Lists/ApplicationNote/Attachments/1910/AN0271V1-NAND%20Error%20Correction%20Codes%20Introduction-0217.pdf`

[7] `connect.ed-diamond.com/auteur/view/58660-patrois_nicolas`

[8] T.K. Moon, *Error Correction Coding – Mathematical Methods and Algorithms*, Wiley Interscience (2005), section 1.5 pour l'identification du signal bruité.

[9] V. Meghdadi, *BER calculation* (2008) à `www.unilim.fr/pages_perso/vahid/notes/ber_awgn.pdf` qui s'inspire de A. Goldsmith, *Wireless Communications* (2004) disponible à `web.cs.ucdavis.edu/~liu/289I/Material/book-goldsmith.pdf`

[10] A. Prodromakis, S. Korkotsides & T. Antonakopoulos, *MLC NAND flash memory : aging effect and chip/channel emulation*, Microprocessors and Microsystems **39**(8) 1052–1062 (2015) à `www.loe.ee.upatras.gr/Comes/Papers/2015_MICPRO%20-%20MLC%20NAND%20Flash%20memory.pdf`

[11] J.-M Friedt, *Radio Data System (RDS) – analyse du canal numérique transmis par les stations radio FM commerciales, introduction aux codes correcteurs d'erreur*, GNU/Linux Magazine France **204** (Mai 2017)

[12] L. Teske, *GOES Satellite Hunt* à `www.teske.net.br/lucas/satcom-projects/satellite-projects/`

[13] C.K.P. Clarke, *Reed-Solomon error correction*, BBC R&D White Paper WHP031 (2002) à `downloads.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP031.pdf`

[14] S.T.J. Fenn, D. Taylor & M. Benaissa, *The design of Reed-Solomon codecs over the dual basis*, Microelectronics Journal **26** 383–391 (1995) et la datasheet du Mitel Semiconductor MS13544 qui implémente les codes de correction convolutifs et par bloc dans un circuit intégré compatible d'applications spatiales (`microelectronics.esa.int/vhdl/doc/MS13544.pdf`)

[15] Y.Guidon, *Comprendre les générateurs de nombres pseudo-aléatoires*, GNU/Linux Magazine France **81** (2006) à `jvf.free.fr/550_Articles_LinuxMag/GLMF_081_064_076.pdf` et Y. Guidon, *Sciences/théorie de l'information : propriétés et dérivés des CRC et LFSR*, GNU/Linux Magazine France **85** (2006) à `jvf.free.fr/550_Articles_LinuxMag/GLMF_085_084_093.pdf`

[16] G.D. Forney, *Concatenated codes* (1965) disponible à `core.ac.uk/download/pdf/4381889.pdf`

[17] A.J. Viterbi, *Wireless digital communication : A view based on three lessons learned*, IEEE Communications Magazine, 29(9), 33–36 (1991) à `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.421.7776&rep=rep1&type=pdf`

[18] D.J. Mudgway, *Uplink-Downlink – A history of the Deep Space Network, 1957–1997*, NASA SP-2001-4227 (2001) à `ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20020033033.pdf`