Réception, décodage et affichage d'une image satellite METEOR-M2

PARTIE 1 : Réception et extraction des données

Mots-clés : Modulation numérique - QPSK - Antennes - Bilan de liaison - Diagramme de constellation.

Niveau concerné :	2ème année BTS Systèmes Numériques (option Informatique et Réseau)
Matière :	Physique
Partie du programme :	- Transmissions numériques sur fréquence porteuse. - Antennes.
Capacités exigibles mobilisées :	- Définir une modulation PSK, et utiliser les signaux en phase i(t) et en quadrature q(t) à partir des données binaires. - Visualiser et interpréter un diagramme de constellation. - Déterminer l'encombrement spectral pour une modulation. - Définir l'impédance d'entrée, le diagramme de rayonnement, le gain, le coefficient PIRE, la polarisation d'une antenne. - Effectuer un bilan de liaison.
Outils utilisés :	- Python 3.8.2 https://www.python.org/downloads/ avec les modules matplotlib et numpy. - Logiciel GNU Radio https://www.gnuradio.org/ (*facultatif : pour la réception en direct uniquement*) - Clé TNT USB (chipset RTL2832u) (de 5 à 40 euros selon les sites et les accessoires fournis). Par exemple : ici . (*facultatif : pour la réception en direct uniquement*) - Jupyter Notebook : Ce TP est facilement réalisable en ouvrant les fichiers "ipynb" avec Jupyter (*facultatif, mais fortement recommandé*)
Fichiers nécessaires fournis :	- « meteor.grc » : fichier pour l'acquisition GNURadio - « binarymeteor.s » et « binarymeteor2.s » : fichiers de données brutes - « viterbi.bin » et « viterbi2.bin » : données brutes décodées par Viterbi - « viterb.py » : fichier contenant des fonctions utiles
Contact :	thomas.lavarenne@ac-creteil.fr
Remerciements :	Cette activité n'aurait pas pu voir le jour sans les travaux passionnants et les explications toujours claires et précises de Jean-Michel Friedt : jmfriedt.free.fr (voir les références [1] et [2] pour approfondir les notions vues ici). Merci également à Daniel Estévez pour les explications et le document du CCSDS.

Réception, décodage et affichage d'une image satellite METEOR-M2

PARTIE 1 : Réception et extraction des données

Meteor-M2 est un satellite météorologique Russe lancé en 2014 et qui se trouve en orbite basse autour de la Terre à une altitude d'environ 830 km. Sa période de révolution est de l'ordre de 100 minutes. Une de ses missions principales est de surveiller le réchauffement climatique en mesurant la température des océans et en estimant la fonte des glaces polaires. Il est équipé de plusieurs capteurs HD qui envoient des données d'imagerie vers les stations aux sols sur 3 bandes principalement. Lorsqu'il transmet en journée, elles se situent généralement dans le visible pour la première, dans le rouge et le proche infrarouge pour la deuxième et dans l'infrarouge plus lointain pour la troisième. Les données images reçues sont envoyées compressées au format JPEG.

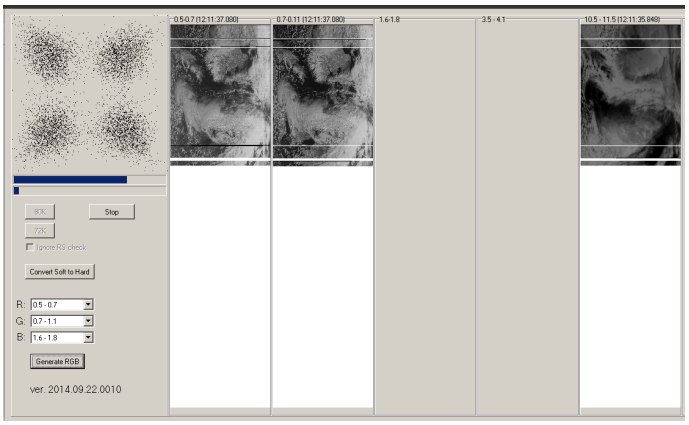


Figure 1 – Le programme "LRPT Offline Decoder" permet de décoder les données reçues et affiche l'image en tenant compte des informations reçues sur les trois canaux.

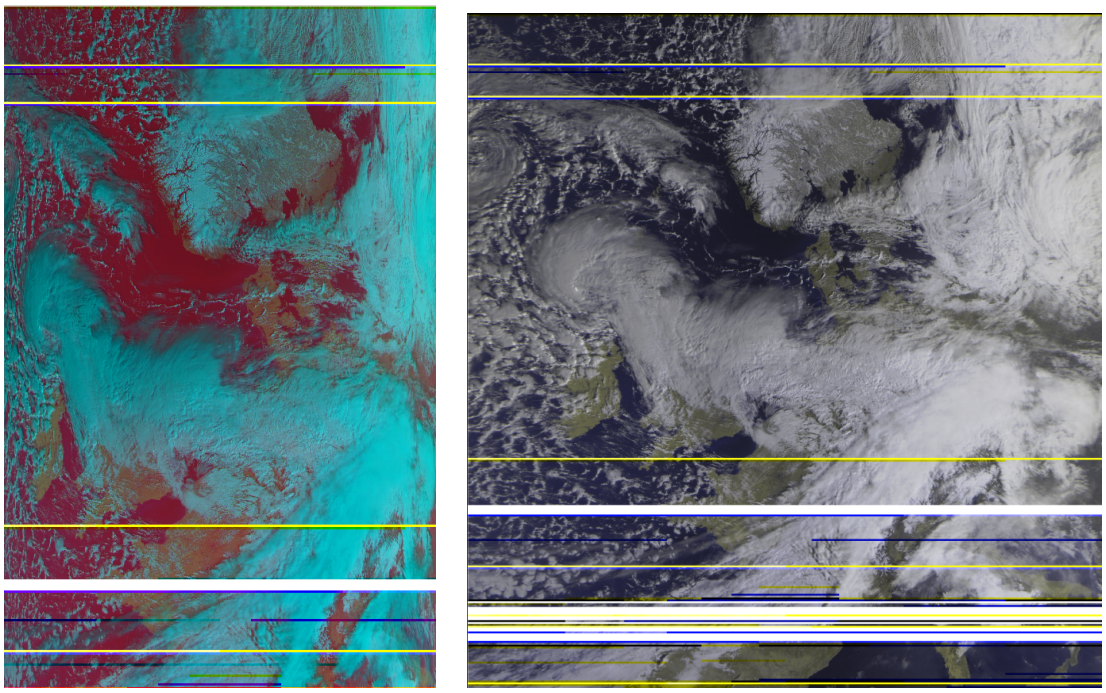


Figure 2 – D'autres programmes comme "LRPT Enhancer" permettent de changer le type de projection ainsi que d'afficher des couleurs plus "naturelles" (image "brute" à gauche et améliorée à droite).

Nous nous proposons, au cours de ce TP, d'étudier la chaîne de traitement du protocole LRPT utilisé ici, **de la réception à l'affichage de l'image**.

I Réception du signal

1) Le protocole LRPT

Il existe différents protocoles utilisés par les différents satellites météorologiques en orbite basse autour de la Terre : APT, LRPT, HRPT...etc. Le satellite Meteor M2 utilise le protocole LRPT, relativement bien documenté.

Doc. 1 : Protocole LRPT ([6])

The Low Rate Picture Transmission (LRPT) is a digital transmission system, intended to deliver images and data from an orbital weather satellite directly to end users via a VHF radio signal. It is used aboard polar-orbiting, near-Earth weather satellite programs such as MetOp (LRPT permanently deactivated on MetOp-A due to interference with on HIRS) and NPOESS (replaced with JPSS), and is currently being used on METEOR-M satellites.

Characteristics :

LRPT transmits at ∼ 120 kHz Bandwidth. The signal transmits using QPSK at 72 kBd (72,000 symbols per second) with a raw data rate of 144 kbps, with an equivalent isotropically radiated power (EIRP) level that varies between 3.2 dBW and 8.0 dBW. The datastream is processed using a Reed–Solomon error correction, then convolution encoded, interleaved, and padded with unique synchronization words.

Each sensor on the satellite that uses LRPT to transmit its data is considered an application and provided a percentage of the transmission bandwidth in the form of a virtual channel.

Frequencies :

Meteor-M N1 137.025 MHz - 137.925 MHz (Reported decommissioned)
Meteor-M N2 137.1 MHz - 137.925 MHz (Active)
Meteor-M N2-1 137.1 MHz - 137.9125 MHz (Launch failure)
Meteor-M N2-2 137.1 MHz - 137.9125 MHz (Failed - as of Jan. 2020)

La fréquence actuelle de transmission est 137,1 MHz, et utilise une onde électromagnétique **polarisée circulairement**.

2) Antenne dipôle en V

Une antenne simple à réaliser et permettant l'obtention de résultats satisfaisants est l'antenne dipôle "en V" (on applique un certain angle entre les deux brins afin de réaliser la meilleure adaptation d'impédance possible).

1. **Calculer la longueur théorique de chaque brin du dipôle (longueur totale du dipôle : $\lambda/2$).**

2. **Pourquoi est-il important d'assurer l'adaptation d'impédance dans le cas de la réception ou de l'émission d'un signal électromagnétique ?**

3. **En utilisant la Figure 3 ci-dessous, déterminer la valeur de l'angle théorique optimal pour assurer l'adaptation d'impédance à appliquer entre les deux brins du dipôle ?**

4. **En utilisant les informations données en introduction de ce document, celles données dans le Doc. 1 et sur la Figure 3, établir un bilan de liaison afin d'estimer la puissance en W absorbée par l'antenne en réception pour les deux valeurs extrêmes de la PIRE donnée. Convertir ces puissances en dBm.**

5. **En estimant la sensibilité du récepteur DVB-T utilisé à -110 dBm, peut-on espérer récupérer un signal significatif grâce à notre dispositif ?**

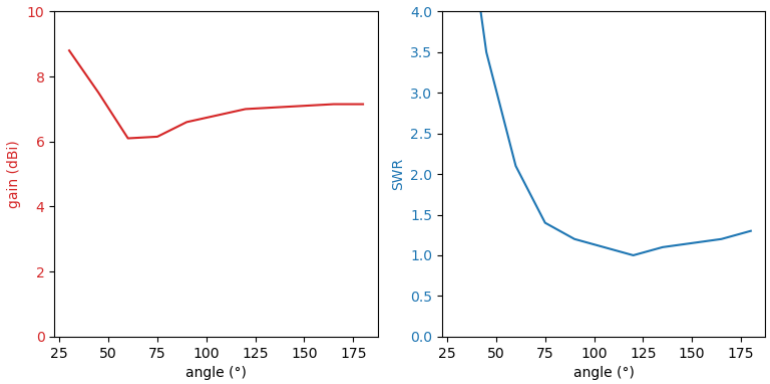


FIGURE 3 – Diagramme issu des données de [7] représentant la variation du gain en dBi (pour une hauteur donnée) et du taux d'onde stationnaire (SWR) en fonction de l'angle donné au dipôle.

Pour affiner un peu notre estimation, il faut tenir compte du fait que l'onde émise par le satellite est **polarisée circulairement**.

6. **Quelle devrait être la polarisation de l'antenne de réception pour récupérer le maximum de signal ? Est-ce le cas ?**

On peut montrer que le fait de réceptionner une onde polarisée circulairement avec une antenne dipôle polarisée rectilignement engendre **une perte de 3 dB** (on ne récupère qu'une seule des deux composantes circulaires incidentes).

7. **Peut-on tout de même espérer une réception convenable du signal ? Pourquoi utiliser une antenne dipôle alors ?**

3) Réception du signal et enregistrement des données

Ouvrir le fichier "meteor.grc" ([5]) représenté sur la Figure 4 et compléter le champ **Frequency**.

8. **Calculer la valeur du paramètre "omega" du bloc "Clock Recovery MM" sachant qu'il correspond au nombre d'échantillons par symbole.**

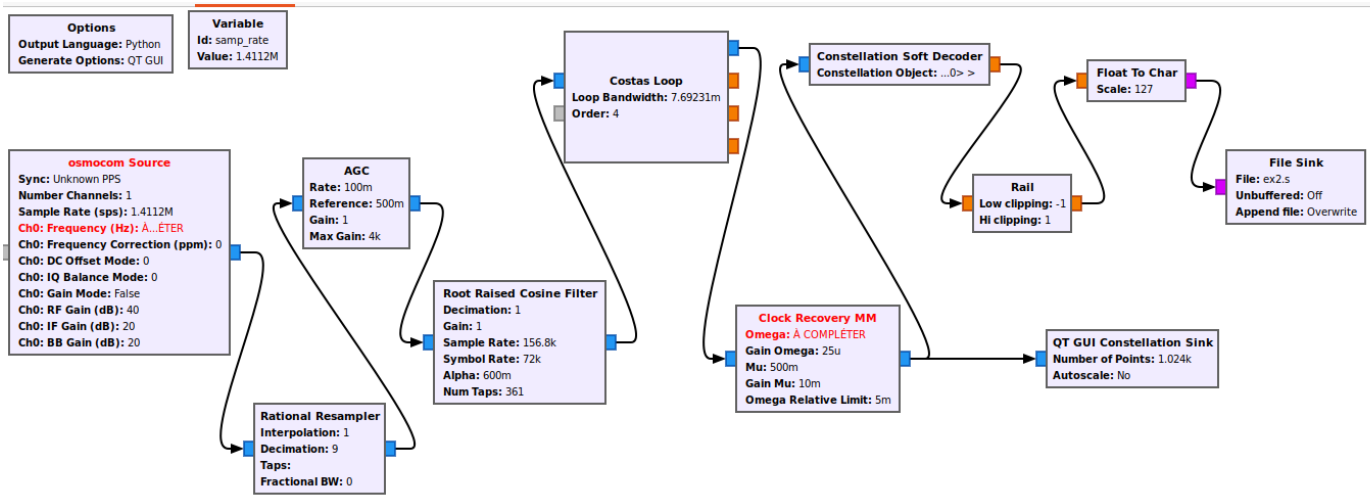
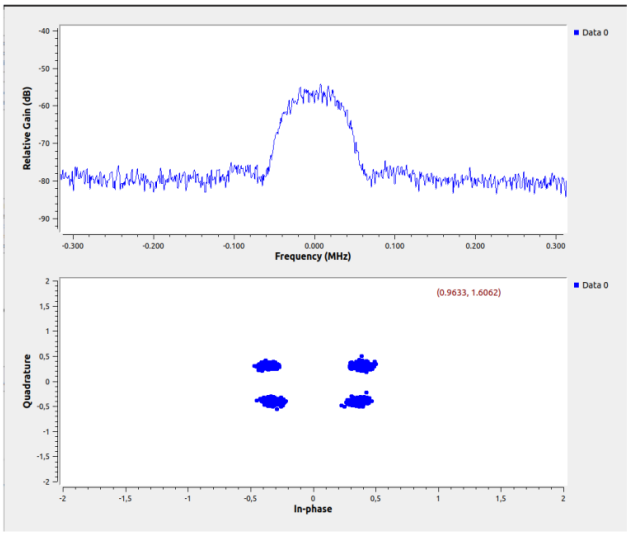


FIGURE 4 – Graphe GNU Radio permettant de réceptionner les échantillons I et Q en sortie du bloc Osmocom Source.

9. **Quel est le type de modulation ? Expliquer rapidement. Combien a t-on de bits par symboles ?**

On rajoute un bloc **"QT GUI Frequency Sink"** directement en sortie du bloc source, on observe les graphiques suivants lors de la transmission :



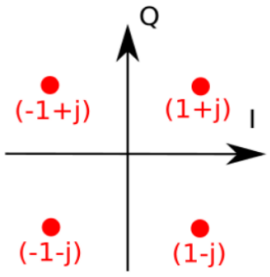


FIGURE 5 – Réponse en fréquence et constellation observées au cours de la réception.

10. **Estimer la valeur de l'occupation spectrale de la modulation QPSK. Comparer aux informations données dans le Doc. 1.**

11. **En observant la Figure 5, à combien de dB au-dessus du bruit se situe le signal ? Est-ce cohérent avec le bilan de liaison effectué plus haut ?**

12. **Observer les paramètres du bloc "Constellation Soft Decoder" et compléter le schéma (Figure 5 à droite) avec les valeurs binaires 00, 01, 10 et 11 correspondant à chaque points de coordonnées $\pm 1 \pm j$**

digital.constellation_calcdist(([-1-1j, -1+1j, 1+1j, 1-1j]), ([0, 1, 3, 2]), 4, 1).base()

Doc 2 : Constellation Soft Decoder (*d'après* [2])

En fonction des position du point dans le plan complexe, le bloc renvoie deux échantillons successifs correspondant aux coordonnées de ce point comme le montre la figure ci-dessous :

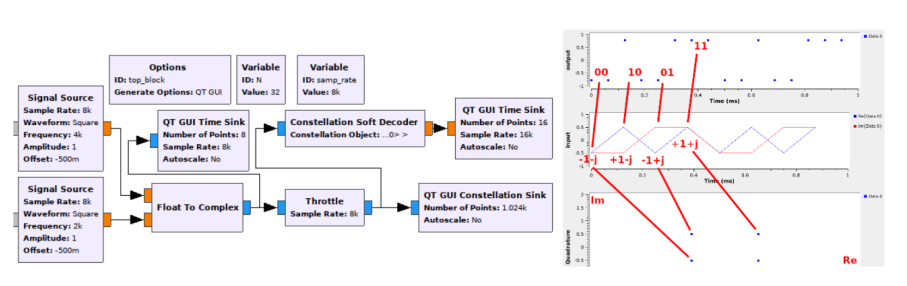


FIGURE 6 – Représentation dans le plan complexe des échantillons (constellation) et représentation temporelle en sortie du bloc Constellation Soft Decoder (issue de [2] p2).

Pour le traitement ultérieur, chaque valeur comprise entre [-1 ;1] est convertie dans la gamme [-127 ;127] et stockée sur un octet (signed char) dans un fichier binaire nommé ici "ex.s". Par la suite nous nommerons "**softs bits**" ces octets.

C'est ce fichier qui est traité par "LRPT Offline Decoder" présenté en introduction ou encore par "meteor-decoder" ([4]) et c'est ce fichier que nous allons traiter pour afficher l'image JPEG.

Pour simplifier et diminuer la quantité de donnée à traiter, nous allons travailler sur une petite fraction de ce fichier (quelques lignes correspondant au milieu de l'image). Ce fichier de travail se nomme "binarymeteor.s".

II Des "softs bits" aux bits

1) Affichage des "softs bits" et de la constellation

Se placer dans le dossier de travail et ouvrir un terminal python3, importer la bibliothèque numpy, ouvrir le fichier binaire et lire les valeurs avec le type int8 :

```
>>> import numpy as np
>>> f=open("binarymeteor.s","rb")
>>> soft_bits = np.fromfile(f, dtype=np.int8)
```

13. **Faire afficher les 20 premières valeurs pour vérifier le bon fonctionnement de vos commandes.**

Afin de valider complètement notre compréhension de ce fichier, nous allons maintenant ré-afficher la constellation avec matplotlib. Pour cela, récupérer les valeurs **paires** dans une nouvelle liste x (abscisses) et les valeurs **impaires** dans une nouvelle liste y (ordonnée). Taper les commandes ci-dessous en complétant éventuellement les pointillés :

```
>>> import matplotlib.pyplot as plt
>>> x=[]
>>> y=[]
>>> for i in range(len(soft_bits)//2):
...   x.append(soft_bits[....]/127.0)
...   y.append(soft_bits[....]/127.0)

>>> plt.plot(x,y,".")
>>> plt.show()
```

14. **Afficher le graphe. Quel est le rôle de la division par 127 ?**

15. **Proposer une méthode pour obtenir le message binaire (passage des "softs bit" aux "hards bits" (0 ou 1).**

Compléter le programme permettant de réaliser cette opération et le réaliser.

```
>>> hard_bits=""
>>> for i in range(len(soft_bits)):
...   if soft_bits[i] >= 0:
...     hard_bits += "..."
...   if soft_bits[i] < 0:
...     hard_bits += "..."
```

Vérifier en affichant les 10 premières valeurs de soft_bits et de hard_bits :

Avons-nous terminé le travail de décodage ? Les informations binaires sont-elles exploitables ? Pour le vérifier, il va falloir chercher le code de synchronisation **"1ACFFC1D"** que l'on doit retrouver au début de chaque paquet envoyé (*d'après la documentation*).

2) Mot de synchronisation et algorithme de Viterbi

La documentation technique du protocole LRPT nous informe effectivement que toutes les transmissions sont synchronisées avec le mot 0x**1ACFFC1D**. C'est donc ce mot que nous allons chercher à trouver parmi nos données. Seulement nous n'allons pas le trouver comme ça. La documentation nous indique également que le protocole LRPT, afin de fiabiliser la communication, double le nombre de bits envoyé en utilisant un **encodage de viterbi**. C'est donc l'encodage par viterbi du mot 0x**1ACFFC1D** que nous devons trouver.

Doc 4 : encodage convolutif de viterbi - d'après wikipedia et [8]

L'algorithme de Viterbi, d'Andrew Viterbi, permet de corriger, dans une certaine mesure, les erreurs survenues lors d'une transmission à travers un canal bruité.

Son utilisation s'appuie sur la connaissance du canal bruité, c'est-à-dire la probabilité qu'une information ait été modifiée en une autre, et permet de simplifier radicalement la complexité de la recherche du message d'origine le plus probable.

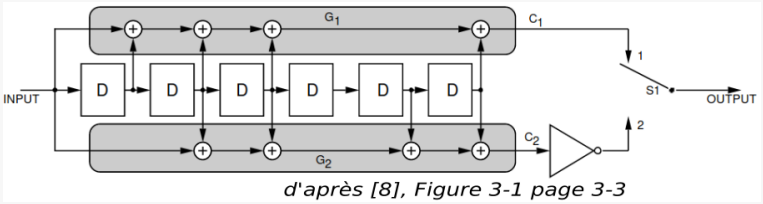
Cet algorithme a pour but de trouver la séquence d'états la plus probable ayant produit la séquence mesurée.

Il s'appuie sur un registre à décalage servant de mémoire, alimenté par le nouveau bit de la séquence à encoder, et alimentant une ou plusieurs portes XOR pour fournir ici 2 bits en sortie pour chaque bit d'entrée.

Pour cela il utilise deux polynômes générateurs nommés G1 et G2 et qui valent ici :

G1=[1,1,1,1,0,0,1] #0x6D

G2=[1,0,1,1,0,1,1] #0x4F



d'après [8], Figure 3-1 page 3-3

16. **La fonction *encode_viterbi(mot,G1,G2)* permet d'encoder un mot binaire selon l'encodage convolutif décrit ci-dessus. Compléter les pointillés pour trouver l'équivalent du mot de synchronisation encodé par l'algorithme de viterbi :**

```
>>> import viterb
>>> G1=[1, 1, 1, 1, 0, 0, 1]
>>> G2=[1, 0, 1, 1, 0, 1, 1]
>>> mot = [0, 0, 0 ..................................... , 0, 1]
>>> viterb.encode_viterbi(mot,G1,G2)
```

17. **Chercher dans la suite hard_bits si la séquence obtenue ci-dessus est présente, et si oui, combien de fois. Commenter.**

```
>>> hard_bits.count(".....")
```

Pour la démodulation, le récepteur possède un oscillateur local (OL) qui doit recréer une copie de la fréquence porteuse qui a été générée par l'oscillateur au niveau de l'émetteur (satellite). C'est en mélangeant la fréquence recréée par l'oscillateur local avec le signal réceptionné que la démodulation a lieu. Il n'y a cependant aucune raison que les deux oscillateurs soient parfaitement synchronisés et ce décalage en fréquence δf entre les deux oscillateurs, produit un déphasage qui augmente ou diminue continuellement. Si aucun dispositif n'est mis en place pour compenser ce décalage, cet effet va se traduire par une rotation continue des symboles dans le plan (I,Q). Pour éviter cela, on utilise la **boucle de Costas** ("Costas loop" dans le fichier grc), qui est un système à verrouillage de phase (PLL - *Phase Loop Lock en anglais*), comme le montre la Figure 7 ci-dessous dans laquelle la boucle de Costas a été désactivée.

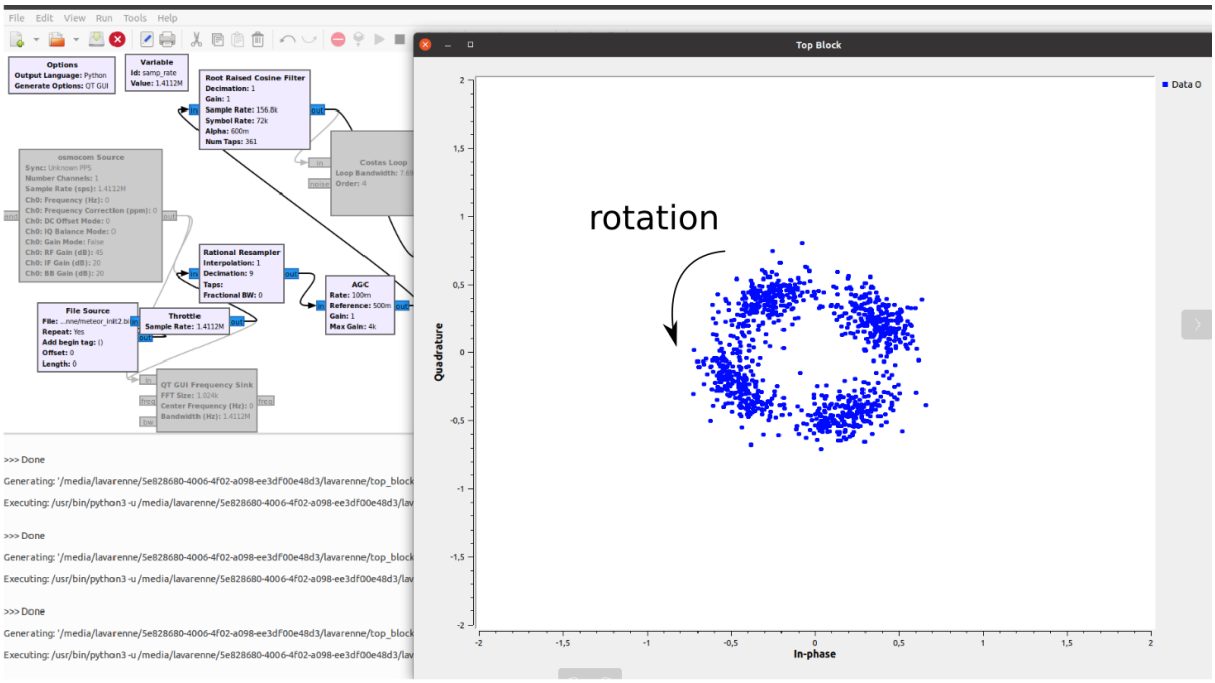


FIGURE 7 – La boucle de Costas permet de compenser le décalage en phase généré lors de la réception du signal. Sa désactivation lors de la réception produit une rotation continue des symboles de la constellation, le sens de rotation dépendant du signe de δf.

18. **Proposer une ou plusieurs explications au résultat de la question précédente.**

3) Rotations au sein de la constellation

Pour résoudre notre problème, il faut lever l'ambiguïté sur la valeur de la phase. On peut alors montrer qu'il y a, par permutation circulaire, 4 possibilités qui suivent le code de Gray :

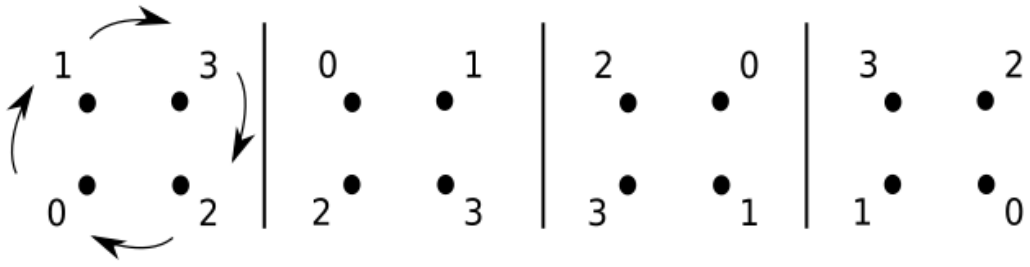


FIGURE 8 – Rotations et verrouillages possibles de la constellation.

Le code de synchronisation précédent n'est donc peut-être pas sous la forme que nous avons vue précédemment. Il pourrait être, selon les quatre possibilités ci-dessous :

1 : 01010110000010000001110010010111000110101010011100111110100111110

2 : 00000011101011101000011011000001100011111111000110010100100010111

3 : 10101001111101111110001101101000111001010101100011000001011000001

4 : 11111100010100010111100100111110011100000000111001101011011010000

Nous allons à présent utiliser une fonction de numpy qui va réaliser la corrélation entre les différents mots possibles et les valeur des softs bits. Lorsque nous affichons la partie réelle de cette corrélation, la présence de pics de corrélation positifs indique la présence du motif ou de son complémentaire (pics de corrélation négatifs) dans le fichier étudié :

```
>>> mot = np.array([[0,1,0,1,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,0,0,1,0,0,1,0,1,1,1,
                                                  0,0,0,1,1,0,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,
                                                  1,0,1,0,0,1,1,1,1,1,1,0],
[0,0,0,0,0,0,1,1,1,0,1,0,1,1,1,0,1,0,0,0,0,1,1,0,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,
                                                  1,1,0,0,0,1,1,0,0,1,0,1,0,0,1,0,0,1,0,1,1,
                                                  1],
[1,0,1,0,1,0,0,1,1,1,1,1,0,1,1,1,1,1,1,0,0,0,1,1,0,1,1,0,1,0,0,0,1,1,1,0,0,1,0,1,0,1,
                                                  0,1,1,0,0,0,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,
                                                  1],
[1,1,1,1,1,1,0,0,0,1,0,1,0,0,0,1,0,1,1,1,1,0,0,1,0,0,1,1,1,1,1,0,0,1,1,1,0,0,0,0,0,0,
                                                  0,0,1,1,1,0,0,1,1,0,1,0,1,1,0,1,1,0,1,0,0,
                                                  0]])
>>> for i in range(len(mot)):
>>> ...    x = np.correlate(soft_bits, mot[i]-np.mean(mot[i]))
>>> ...    plt.figure(i+1)
>>> ...    plt.plot((x.real))
>>> plt.show()
```

19. **Le motif cherché est-il présent ? Noter le numéro de la figure présentant des pics de corrélation positifs.**

20. **Combien compte t-on de corrélations ? De quelle valeur chaque pic est espacé du suivant ? Est-ce cohérent avec la valeur de 16384 échantillons annoncée dans la documentation ainsi qu'avec la taille totale du fichier en octet ?**

4) Silence.. On tourne !

21. **Pour remettre les symboles "à leurs places" dans la constellation, quelle(s) opération(s) faut-il effectuer avec les hard bits ? et avec les soft bits ?**

22. **Compléter le programme suivant permettant de remettre les symboles "dans l'ordre" :**

```
>>> for i in range(0,len(soft_bits)):
>>> ...
>>> g=open("entreeViterbi.bin","wb")
>>> g.write(soft_bits)
>>> g.close()
```

23. **Reprendre les questions 15 et 17 (conversion en hard_bits puis recherche du mot de synchronisation encodé par Viterbi). Est-ce bon cette fois-ci ? Commenter.**

5) Viterbi intervient !

GNU Octave

La fonction **decode_viterbi(mot)** permet de décoder un mot binaire qui a préalablement été encodé par l'algorithme de Viterbi en renvoyant le nombre d'erreur et le résultat traduit en hexadécimal. Cette fonction fait intervenir une fonction GNU Octave contenue dans le fichier "viterbi.m". Afin d'éviter l'installation un peu lourde du programme GNU Octave et des modules "signal", "control" et "communications" nécéssaires au bon fonctionnement de ce script, observer attentivement le gif animé "**viterbi.gif**" qui se trouve dans le dossier "images" (images/viterbi.gif) afin de répondre aux deux questions suivantes et appréhender ainsi la puissance et l'efficacité de ce type d'encodage.

Nous avons vu qu'il était possible d'encoder une suite de bits avec l'algorithme de viterbi, voyons maintenant quel est son intérêt et comment faire le chemin inverse. Commençons par traduire notre mot

01010110000010000001110010010111000110101010011100111101001111 10

Nous devrions retrouver 0x1ACFFC1D..

```
>>> mot="01010110000010000001110010010111000110101010011100111101001111 10"
>>> error, decode=viterb.decode_viterbi(mot)
>>> print(error)
>>> print(decode)
```

```
>>> mot="01010110000010000001110010010111000110101010011100111101001111 10"
>>> error, decode=viterb.decode_viterbi(mot)
>>> print(decode)
1ACFFC1D

>>> print(error)
0
```

24. **Inverser 1 bit au hasard dans le mot précédent et recommencer le décodage**

25. **Inverser plusieurs bits au hasard dans le mot précédent et recommencer le décodage**

Conclusion : Ne pas négliger viterbi, il a un rôle crucial ! La plupart des communications numériques actuelles mettent en jeu l'algorithme de Viterbi associé à d'autres codes correcteurs d'erreurs (Reed-Solomon par exemple). On le retrouve par exemple dans les protocoles utilisés par la TNT, la TV par satellite, l'ADSL...etc. Sans ces codes correcteurs, la communication serait impossible (en tout cas très limité et au voisinage proche de l'émetteur).

6) décodage des softs bits par l'algorithme de viterbi

Nous venons de voir qu'il est crucial de traiter nos données brutes par l'algorithme (inverse) de Viterbi afin d'éliminer les erreurs dues à la transmission. Il existe une librairie écrite en c++ qui permet de décoder les softs bits en utilisant l'algorithme de viterbi (https ://github.com/quiet/libfec). À titre indicatif et pour ceux qui souhaiteraient reproduire cette étape, ci-dessous le programme que nous avons utilisé afin d'obtenir le fichier décodé (fortement inspiré de [1]). Cette étape à été réalisée avec le fichier **binarymeteor2.s** et le résultat du décodage se trouve dans le fichier **viterbi2.bin**.

```c
// adaptation du script utilisé par JM Friedt
// gcc -o jmf libfec jmf libfec.c -I./libfec ./libfec/libfec.a
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h> // read
#include <fec.h>
#define MAXBYTES (819200/16)  // Indiquer taille en octet du fichier à traiter /16
#define VITPOLYA 0x4F
#define VITPOLYB 0x6D
int viterbiPolynomial[2] = {VITPOLYA, VITPOLYB};
int main(int argc,char *argv[]){
  int i,framebits,fd;
  unsigned char data[MAXBYTES],*symbols; // [8*2*(MAXBYTES+6)]; // *8 for bytes->bits & *2 Viterbi
  void *vp;
  symbols=(unsigned char*)malloc(8*2*(MAXBYTES+6));
  fd=open("./entreeViterbi.bin",O RDONLY); read(fd,symbols,MAXBYTES*16); close(fd);
    for (i=0;i<MAXBYTES*16;i+=1) symbols[i]=-symbols[i];    //libfec require symbols inversion (uint8 t and int8 t conversion)
    for (i=0;i<MAXBYTES*16;i+=2) symbols[i+1]=-symbols[i+1];    //Q inversion (most popular convention used by meteor)
  framebits = MAXBYTES*8;
  set viterbi27 polynomial(viterbiPolynomial);
  vp=create viterbi27(framebits);
  init viterbi27(vp,0);
  update viterbi27 blk(vp,symbols,framebits);
  chainback viterbi27(vp,data,framebits,0);
  fd=open("./sortieViterbi.bin",O WRONLY|O CREAT,S IRWXU|S IRWXG|S IRWXO);
  write(fd,data,framebits);
  close(fd);
  exit(0);}
```

26. **Ouvrir avec python le fichier de sortie appelé ici viterbi2.bin et convertir les valeurs en binaire sur 8 bits à l'aide de l'exemple ci-dessous. Conclure**

```
>>> bin(10)
'0b1010'
>>> bin(10)[2:]
'1010'
>>> bin(10)[2:].zfill(8)
'00001010'
>>>
```

```
>>> f=open("......","rb")
>>> bits = np.fromfile(f, dtype=np.uint8)
>>> binary=''
>>> for i in range(len(bits)):
...   binary+=.......
>>> binary[:10*8]
```

27. **Compter le nombre de fois que le mot de synchronisation 1acffc1d (converti en binaire sur 8 bits) se répète dans la liste. Conclure.**

```
>>> binary.count("000......")
```

Références :

[1] J.-M Friedt, *Décodage d'images numériques issues de satellites météorologiques en orbite basse : le protocole LRPT de Meteor-M2 (partie 1/2)*, janvier 2019, http ://jmfriedt.free.fr/glmf_meteor1.pdf

[2] J.-M Friedt, *Décodage d'images numériques issues de satellites météorologiques en orbite basse : le protocole LRPT de Meteor-M2 (partie 2/2)*, mars 2019, http ://jmfriedt.free.fr/glmf_meteor1.pdf

[3] Viterbi decoder with Octave, https ://github.com/Filios92/Viterbi-Decoder/blob/master/viterbi.m

[4] Meteor-decoder, https ://github.com/artlav/meteor_decoder

[5] GNU Radio scripts for METEOR M2 Satellite LRPT reception with AIRSPY SDR, https ://github.com/otti-soft/meteor-m2-lrpt

[6] https ://www.sigidwiki.com/wiki/Low_Rate_Picture_Transmission_(LRPT)

[7] Dick Reid, KK4OBI at QSL, *What Happens If... A Dipole is Bent Sideways into a "V"?*, https ://www.qsl.net/kk4obi/Center-fed%20V-dipoles%20Lateral.html

[8] CCSDS The Consultative Committee for Space Data System, *Recommendation for Space Data System Standards,* https ://public.ccsds.org/Pubs/131x0b3e1.pdf