Communication ADSB
Rapport

EIRBALLON

Élèves :
Mathieu Lacabanne
Ayoub Rachik

Enseignants :
Bertrand Le Gal

Année universitaire 2020-2021

Bordeaux INP
ENSEIRB
MATMECA

Table des matières

1 Introduction

Le projet Eirballoon fait partie du projet NANOSTAR et du 100e anniversaire de l'ENSEIRB-MATMECA qui est partenaire du consortium NANOSTAR.

Une équipe composée de cinq étudiants de l'école d'ingénieurs ENSEIRB-MATMECA ont lancé un ballon-sonde pour prendre des photos de Bordeaux depuis le ciel et mesurer les propriétés de l'atmosphère telles que la température, la pression ou le niveau de rayons UV. Tous ces paramètres ont été envoyés au sol en temps réel à l'aide du protocole LoRa et affichés l'avancée sur les réseaux sociaux. Le ballon a atteint environ 30 km d'altitude ce qui a permis de tester le matériel (y compris le module Lora) dans des conditions spatiales avec la station au sol développée dans le cadre du projet NANOSTAR.

FIGURE 1 – Projet EIRBALLON

Notre objectif lors de ce projet est de concevoir un système embarqué capable de prendre et transmettre des images en protocole ADSB (*Automatic Dependent Surveillance-Broadcast*). Nous pouvons alors séparer notre système en deux parties : **émetteur** et **récepteur**.

Notre système complet devra donc être capable d'effectuer différentes étapes :

— 1 -> Prendre une photo.
— 2 -> Emission de l'image encodée en trames ADSB.
— 3 -> Réception des trames ADSB.
— 4 -> Reconstruction et affichage de l'image à l'aide des trames reçues.

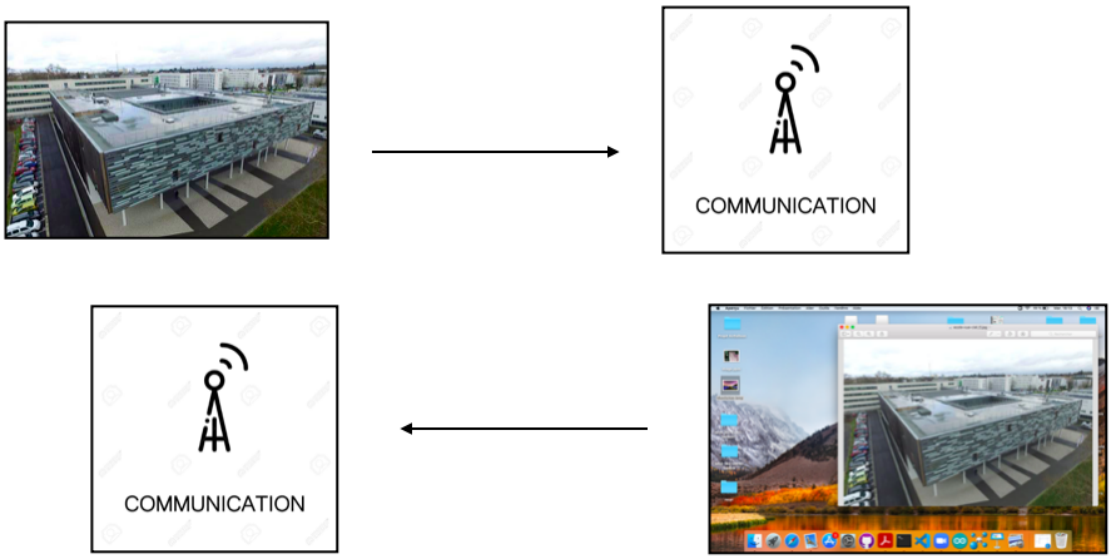

FIGURE 2 – Principe général de notre système

2 Matériels & Configuration

Pour concevoir ce projet, nous avons à notre disposition :
— carte Raspberry Pi : nano-ordinateur monocarte à processeur ARM.
— Camera V2 : module caméra pour Raspberry Pi.
— HackRf One : module radio récepteur-émetteur SDR couvrant de 1MHz jusqu'à 6GHz.

2.1 Partie émission

La partie émetteur est composée d'une carte Raspberry prenant une image RGB à l'aide de la caméra. Puis émet par l'intermédiaire du module radio HackRf des trames ADSB préalablement créés.

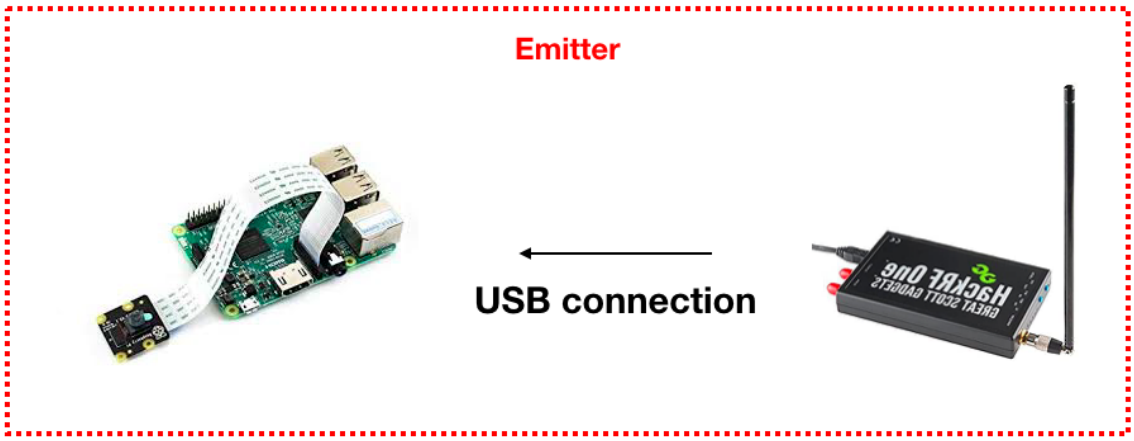

FIGURE 3 – Principe pour émission

2.2 Partie réception

La partie récepteur est composée d'un module radio HackRf qui a pour but de réceptionner les trames ADSB, et d'une carte Raspberry permettant de les traiter pour recomposer l'image RGB émise.

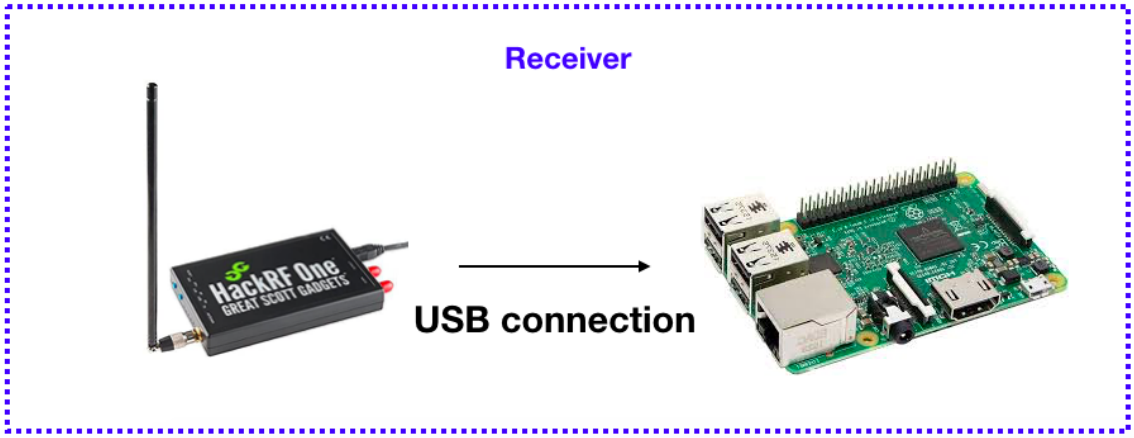

FIGURE 4 – Principe pour réception

2.3 Configurations cartes Raspberry

Tout d'abord, nous avons dû configurer nos cartes Raspberry Pi. Pour cela, nous avons installer une image disque sur une carte mémoire correspondant à la distribution OS voulu sur notre carte (ici la 32 bits). De plus, nous avons configurer notre carte pour pouvoir y accéder à distance avec notre ordinateur en ssh à l'aide de l'application VNC viewer. Pour finir nous y installons les librairies utiles à notre projet comme l'environnement de programmation Visual Code, le compilateur cmake et hackrf pour l'utilisation des modules radio.

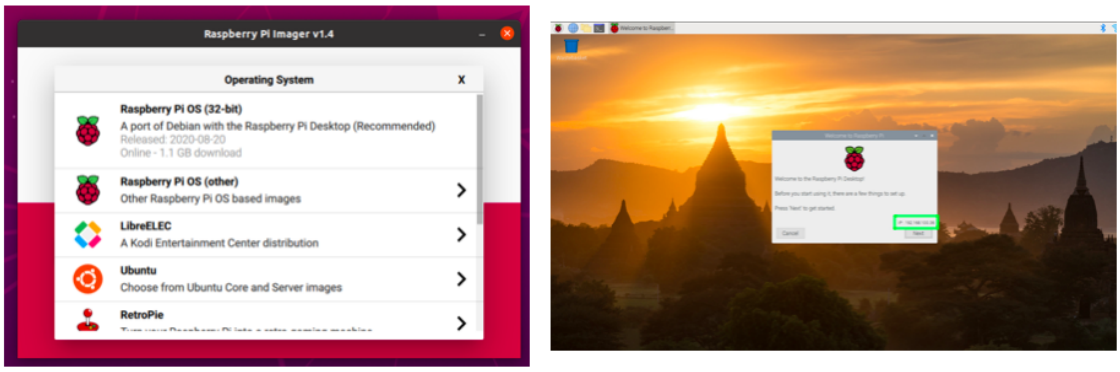

FIGURE 5 – Configuration Raspberry Pi

3 Partie individuelle Mathieu Lacabanne

Dans cette partie, nous nous sommes occupés de la prise de l'image et de la transmission de celle-ci via protocole ADSB.

3.1 Capture

Pour prendre une photo, nous avons écrit une méthode capture capable d'enregistrer une image dans une variable data par l'intermédiaire du module camera. Pour cela, nous utilisons la classe Raspicam utilitaire du module caméra (ici la variable V2 correspond à l'objet Raspicam) permettant d'utiliser les méthodes utiles à la capture de photo en format RGB.

```
1  raspicam::RaspiCam V2;
2  char *data = new char[ V2.getImageTypeSize
       (raspicam::RASPICAM_FORMAT_RGB)];
3
4  //Open camera
5  if ( !V2.open()) {cerr<<"Error opening camera"<<endl;}
6
7  //wait a while until camera stabilizes
8  sleep(3);
9
10 //capture
11 V2.grab();
12
13 //extract the image in rgb format
14 V2.retrieve (data, raspicam::RASPICAM_FORMAT_RGB);
15
16 bmp = new BMP( V2.getWidth(), V2.getHeight(), false);
17 std::copy(data.begin(), data.end(), bmp->data);
```

Listing 1 – Code pour capture photo

FIGURE 6 – Exemple d'une photo prise avec le module camera Raspberry

3.2 Transmission ADSB

3.2.1 Principe de base de la transmission ADSB

Initiallement, le protocole ADSB est utilisé à la surveillance du réseau aérien. Dans ce système, les appareils estiment leurs positions (longitude, latitude, altitude) grâce aux techniques de positionnement par satellite GPS et diffusent ces informations régulièrement. D'un autre côté une tour de contrôle aérienne, a pour objectif de recevoir des données ADS-B et d'afficher en temps-réel la localisation des avions ayant émis des trames ADS-B.

Le protocole ADS-B est utilisé à l'international et permet aux acteurs du réseau aérien d'échanger des données entre eux. Ce protocole obéit à des règles de normalisation pour faciliter le décodage. L'information dans les trames ADS-B est modulé en position d'amplitude (modulation PPM : Pulse Position Modulation). La modulation ici est binaire et est effectuée avec un temps symbole de 1 us. La durée d'une trame est fixe : 120us. Il y a donc 120 symboles dans chaque trame.

La trame est constituée des champs suivants :

— le préambule.
— le format de la voie descendante.
— la capacité.
— l'adresse OACI de l'appareil.
— les données ADS-B.
— les bits de contrôle de parité.

Bit	No. bits	Abbreviation	Information
1-5	5	DF	Downlink Format
6-8	3	CA	Transponder capability
9-32	24	ICAO	ICAO aircraft address
33-88	56	ME	Message, extended squitter
(33-37)	(5)	(TC)	(Type code)
89-112	24	PI	Parity/Interrogator ID

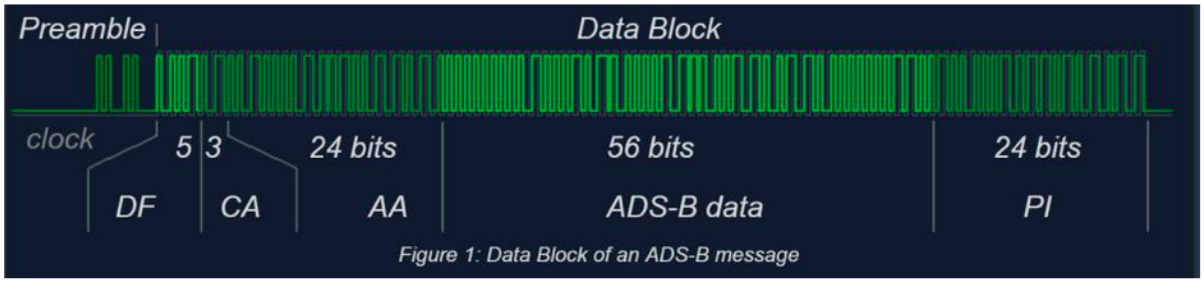

FIGURE 7 – Format d'une trame ADSB

3.2.2 Nouveau format de trame ADSB

Un problème survient à nous ! Le format d'une trame ADSB tel que pour la surveillance aérienne n'est pas du tout adapté pour notre application, nous devons donc créer un nouveau format de trame utile pour notre application. Après réflexion nous créons le format de trames suivant :

— le préambule.
— le type d'information émise.
— l'id : numéro de trames.
— les données émises : pixels de l'image.
— les bits de contrôle de parité = CRC.

Bit	No bits	Abbrevation	Information
1-8	8	ST	Type Frame
8-16	8	ID	Id Frame
16-(8*n+16)	n (1-255)	ME	Message
-	32	PI	CRC

FIGURE 8 – Format d'une trame ADSB

Nous créons donc une classe **Frame** correspondant à ce format de trames.

```
1 class Frame{
2 protected:
3     vector<uint8_t> header_v;
4     vector<uint8_t> config_v;
5     vector<uint8_t> payload_v;
6     vector<uint8_t> crc_field_v;
```

Listing 2 – Code Frame format

Le premier de ces champs permet d'identifier le début d'un message de type ADSB. Il s'agit d'un préambule constant qui permet au receveur de trame ADSB de reconnaître le début d'une trame.

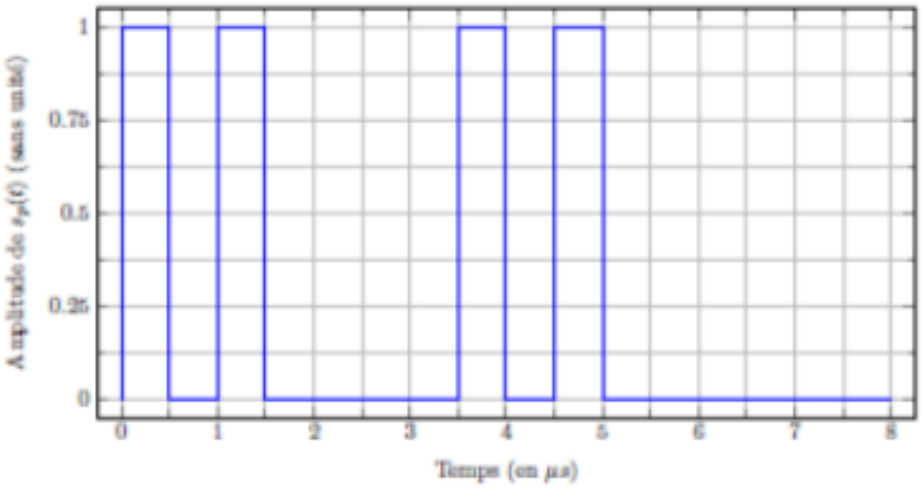

FIGURE 9 – Format du préambule

```
1 header_v.push_back( 1 );   // 1
2 header_v.push_back( 1 );   // 2
3 header_v.push_back( 2 );   // 3
4 header_v.push_back( 0 );   // 4
5 header_v.push_back( 0 );   // 5
6 header_v.push_back( 2 );   // 6
7 header_v.push_back( 2 );   // 7
8 header_v.push_back( 2 );   // 8
```

Listing 3 – Code Préambule

Pour transmettre une image nous devons découper ses données en plusieurs trames. Nous effectuons ce découpage en fonction de la largeur de l'image, une image est constituée de plusieurs ligne de pixels où chaque pixel est décrit en 3 composantes/octets Red Blue Green. Chaque ligne de pixels de l'image est découpée en plusieurs sections que nous allons émettre séparément. Puis, lors de la réception de ces trames, nous reconstituerons l'image à l'aide des données contenu dans les trames ADSB.

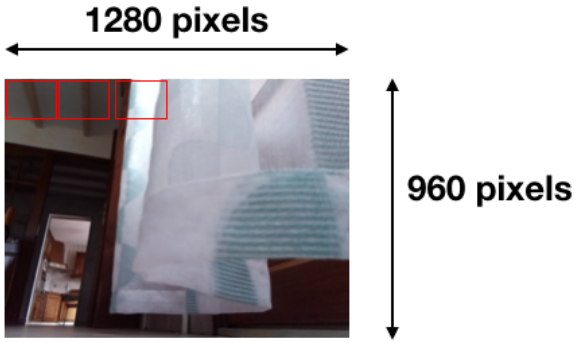

FIGURE 10 – Message transmis dans les trames

Le type Frame permet de définir l'information émise par la trame :
— NEWIMAGE : correspondant à la trame de début d'une image.
— NEWLINE : correspondant à la trame de début de ligne d'une image.
— ENDLINE : correspondant à la trame de fin de ligne d'une image.
— ENDIMAGE : correspondant à la trame de fin d'une image.
— INFOS : correspondant à la trame de données de pixel.

```
1 #define FRAME_INFOS              0xF0 // 1111 0000
2 #define FRAME_NEW_IMAGE          0x0F // 0000 1111
3 #define FRAME_END_IMAGE          0xCC // 1100 1100
4 #define FRAME_NEW_LINE           0x33 // 0011 0011
5 #define FRAME_END_LINE           0xAA // 1010 1010
6 #define FRAME_EMPTY              0x99 // 1001 1001
7
8 #define str_FRAME_INFOS            "FRAME_INFOS    "
9 #define str_FRAME_NEW_IMAGE        "FRAME_NEW_IMG  "
10 #define str_FRAME_END_IMAGE       "FRAME_END_IMG  "
11 #define str_FRAME_NEW_LINE        "FRAME_NEW_LINE"
12 #define str_FRAME_END_LINE        "FRAME_END_LINE"
13 #define str_FRAME_EMPTY           "FRAME_EMPTY    "
```

Listing 4 – Type Frame

L'Id Frame est un numéro identificateur correspondant à l'emplacement de la section émise sur une ligne de l'image. Il nous permet donc de savoir lorsque une trame a été perdu, et de pouvoir reconstituer l'image la plus authentique.

FIGURE 11 – Affichage terminal des trames reçu

Codé sur les derniers 32 bits de la trame, Le calcul de CRC permet au récepteur d'un message de vérifier que les données transmises ne contiennent pas d'erreurs. Pour faire cela, l'émetteur du message calcule une valeur qui est fonction du contenu du message, puis l'ajoute à la fin du message. Le récepteur fait le même calcul, et contrôle que le résultat a la même valeur que celui de l'émetteur.

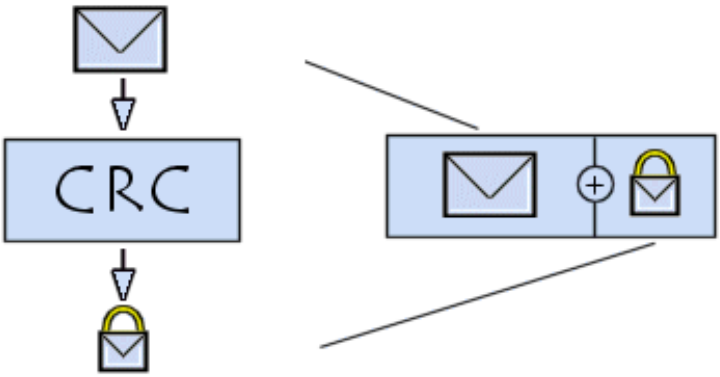

FIGURE 12 – Principe CRC

3.3 Tests et validation de transmission

Pour pouvoir émettre nos trames ADSB préalablement créer encodant les données de l'image, nous utilisons un module radio HackRF. Il faut donc moduler les trames binaires du buffer **buff_1** pour pouvoir les émettre avec le format de donnée utile au module radio.

```
1     while( isFinished == false )
2     {
3         source->execute( &f ); // On fill les donnees
              de la trames avec des donn es de l'image
4         isFinished = !source->is_alive();
5
6         f.compute_crc();
7
8         if( verbose == true )
9         {
10            f.dump_frame();
11        }
12
13        f.get_frame_bits( buff_1 );
14        ppm.execute      ( buff_1, buff_2 );
15        up.execute       ( buff_2, buff_3 );
16        iqi.execute      ( buff_3, buff_4 );
17        radio->emission  ( buff_4 );
18
19        usleep( sleep_time );
20
21        nFrames += 1;
22
23    }
```

Listing 5 – Modulation Trame

Néamoins au cours du projet de nombreux problèmes de transmission sont survenus, principalement de la perte de trame émise, et l'élaboration de la transmission d'une image sans perte a été longue et fastitieuse. La correction de la transmission de trame ADSB avec l'utilisation des modules radios a été établie par Monsieur Le Gal.

3.3.1 Test sans module radio

D'abord, nous avons réalisé des tests sans les modules radios pour ne pas avoir les problèmes de transmission. Nous simulons la transmission en écrivant nos trames ADSB décrivant notre image dans un fichier .RAW. Coté réception, nous re-lisons ce fichier et recomposons l'image d'après les données émis dans les trames.

FIGURE 13 – Test sans module radio

3.3.2 *Test avec module radio*

Puis nous avons réitérés nos tests avec les modules radios. Comme décrit précédemment, lors de la transmission radio, de nombreuses trames sont perdues et dégrade donc l'image coté réception.

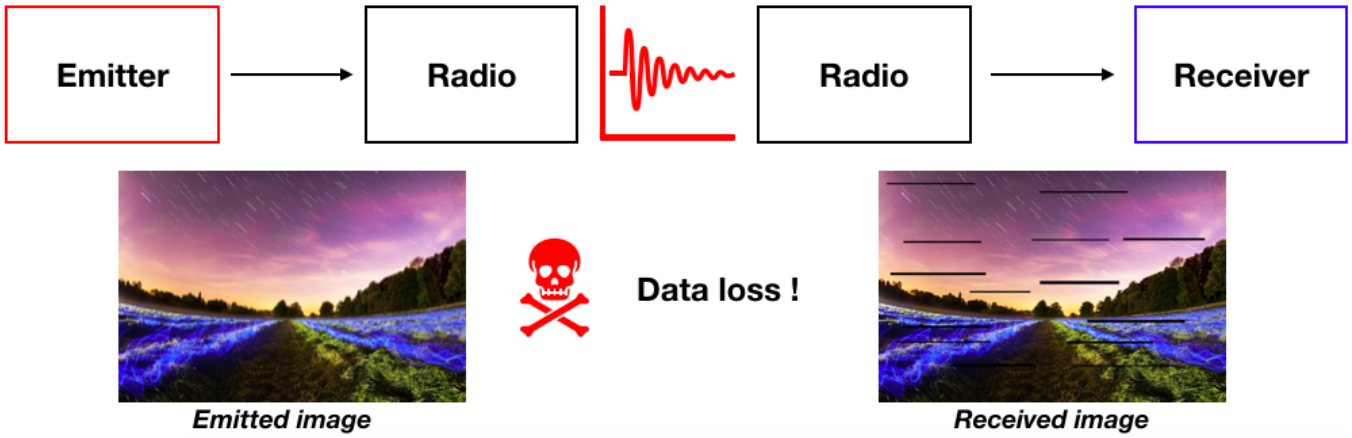

FIGURE 14 – Test avec module radio

3.4 Amélioration de la transmission

Pour améliorer la transmission nous avons pensé à rajouter des concepts permettant de corriger les erreurs de transmission.

3.4.1 Redondance

Tout d'abord nous avons pensé à rajouter de la redondance à notre trame. La redondance serait une recopie à la fin de la trame des données de l'image émise et pourrait ainsi pouvoir corriger les éventuelles erreurs de bit de la trame. En effet, il est possible que certaines trames soit perdues car le CRC est faux. Or, le calcul du CRC peut être faux si seulement un bit de la trame reçu est faussé (par la modulation radio). Ainsi, la redondance nous permettrait à l'aide des données reçues de l'image et du CRC de corriger la trame.

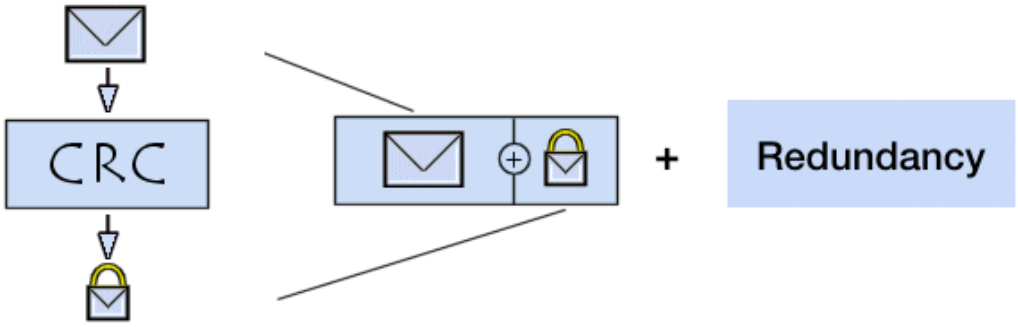

FIGURE 15 – Principe de la redondance

3.4.2 Emission croisée

Puis, nous avons pensé a établir une émission croisée. Au lieu d'envoyer nos trames 1, 2, 3 .. tels quelles, nous entrelaçons les trames pour émettre les trames a, b, .. . Ainsi, lorsque des trames (a, b) sont perdues lors de la transmission et que nous effectuons l'opération inverse pour obtenir les trames 1, 2 .. coté réception, nous pourrons corriger plus facilement les trames incomplètes avec le CRC et la redondance. En effet, lors de la recomposition des trames 1, 2 .. côté réception, nous aurions le nombre de trame décrivant une image voulu avec potentiellement quelques erreurs de bits, alors que sans une émission croisée il est possible de perdre une trame complète qui serait dans ce cas ci impossible à corriger.

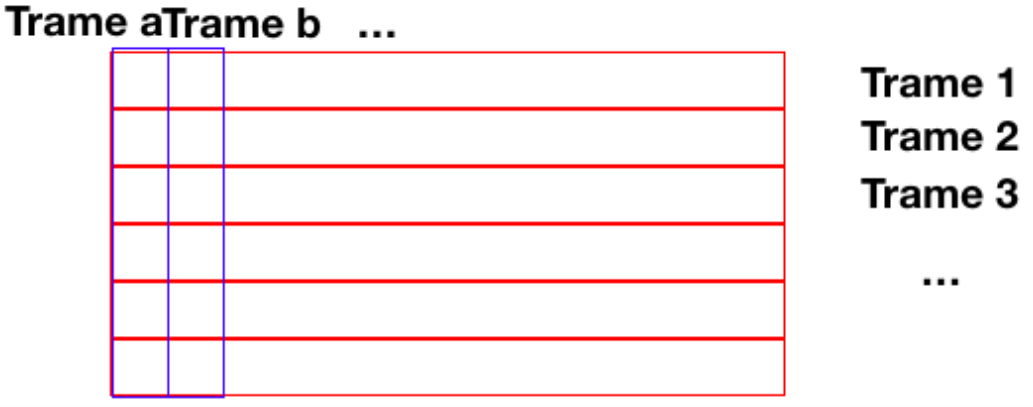

FIGURE 16 – Principe de l'émission croisée

Voici un exemple de code utilisé pour effectuer une émission croisée :

```
1       uint32_t nFrames = 0;
2
3       Frame f2( param.toInt("payload") ); //new frame
            croised emission
4       std::vector<uint8_t> buff_0(
            (source->nbPixel()*3*8/f.payload_size())*8*(f.conf_size()
            + f.payload_size() + f.crc_size()));
            //(source->nbPixel()*3/f.payload_size())*8*(f.conf_size()
            + f.payload_size() + f.crc_size())
5
6       while( isFinished == false )
7       {
8           source->execute( &f ); // On fill les donnees
                de la trames avec des donn es de l'image
9           isFinished = !source->is_alive();
10
11          f.compute_crc();
12
13          f.get_data_bits(buff_0, nFrames);
14
15          nFrames += 1;
16      }
17
18      for(int i = 0; i < 8*(f.conf_size() +
            f.payload_size() + f.crc_size()); i++){
19          buff_1[0] = 1;
20          buff_1[1] = 1;
21          buff_1[2] = 2;
22          buff_1[3] = 0;
23          buff_1[4] = 0;
24          buff_1[5] = 2;
25          buff_1[6] = 2;
26          buff_1[7] = 2;
27          for(int j = 0; j < nFrames; j++){
28              buff_1[j + f.header_size()] = buff_0[i +
                    j*8*(f.conf_size() + f.payload_size() +
                    f.crc_size())];
29          }
30
31          ppm.execute      ( buff_1, buff_2 );
32          up.execute       ( buff_2, buff_3 );
33          iqi.execute      ( buff_3, buff_4 );
34          radio->emission  ( buff_4 );
35
36          usleep( sleep_time );
37      }
```

Listing 6 – Emission croisée

4 Partie individuelle Ayoub Rachik

Dans cette partie, nous nous sommes occupé de l'affichage de l'image une fois réceptionner en protocole ADSB.

4.1 Utilisation de la librairie SFML

SFML est un langage qui fournit une interface simple aux différents composants du PC, pour faciliter le développement de jeux et d'applications multimédias. Il est composé de cinq modules : système, fenêtre, graphique, audio et réseau. Afin d'utiliser ce langage on importe les bibliothèques suivantes :

- SFML/System.hpp : cette bibliothèque permet d'utiliser les utilitaires de SFML notamment les fonctionnalités de temps.
- SFML/Window.hpp : cette bibliothèque nous permet de gérer l'ouverture, la fermeture des bibliothèques et l'affichage.
- SFML/Graphics.hpp : cette bibliothèque nous permet de configurer les paramètres de dessin et de gérer le contenu de l'image à afficher.

FIGURE 17 – SFML

4.2 Affichage d'une image

Afin de réceptionner une image BMP et de l'afficher, nous avons réalisé l'algorithme suivant :

```
sf::RenderWindow window;
        window.clear()
        while (window.pollEvent(event))
                for i from 1 to Width
                        for j from 1 to length
                                sf::Color color(tab);
                                image.setPixel(i, j, color);
        window.close();
```

FIGURE 18 – Affichage et actualisation de l'image

Tout d'abord, nous commençons par ouvrir la fenêtre d'affichage avec la fonction RenderWindow, puis nous répétons la boucle tant qu'il y a des événements à traiter. Ensuite, nous parcourons le fichier texte contenant les pixels en longueur et en largeur, nous définissons ces pixels dans l'image que nous allons montrer. Enfin, nous fermons la fenêtre lorsque nous terminons la boucle while. A cause d'un problème de conversion, vu que sf : :Color color, et image.setPixel ne prennent que des argument en format [R,G,B].

5 Conclusion

Le système ADSB est un protocole de communication qui nous permetttra d'avoir de meilleure performance pour construire la communication du projet Eirbaloon. En effet, par rapport à une communication LoRa, la communication ADSB nous permettra d'avoir un flux de données plus important. Par contre la communication ADSB n'est pas du tout sécuritaire vis à vis des données transmis car tout le monde peut recevoir nos trames émises sur notre bande de fréquence choisie. Néanmoins, ceci n'a pas d'importances pour notre application.

Ce projet nous a permis de pouvoir envoyer et recevoir des trames de données ADSB à l'aide de carte Raspberry et module radio HackRf pour réaliser un émetteur et récepteur ADSB.

Pour améliorer notre système, nous pouvons éviter les pertes de données lors de leurs transmissions. Cela peut être fait en ajoutant un contrôleur d'erreurs CRC plus abouti et en corrigeant les données perdues par redondance ou en compilant des blocs de données croisés.

Figure 19 – EIRBALLON