BASA Camille
BON Cyril
PORMENTE Swen
CUDENNEC Corentin

0 Introduction :

Le cyberphare est un projet de contrôle à distance de lyre afin d'améliorer le cadre de travail du LaserWave lors des évènements dans l'Institut d'Optique Graduate School. Pour permettre cela, nous cherchons d'abord les façons de contrôler le plus facilement une lyre, puis y ajouter des couches supplémentaires que sont le stroboscope, la liaison sans fil etc. L'ergonomie, de part la disposition des boutons et potentiomètres est aussi prise en compte dans le cahier des charges afin de manipuler les lyres facilement, n'étant cependant qu'un prototype, la forme que prendra la télécommande finale restera assez basique.

1 Conception :

1.1 réalisation du cahier des charges :

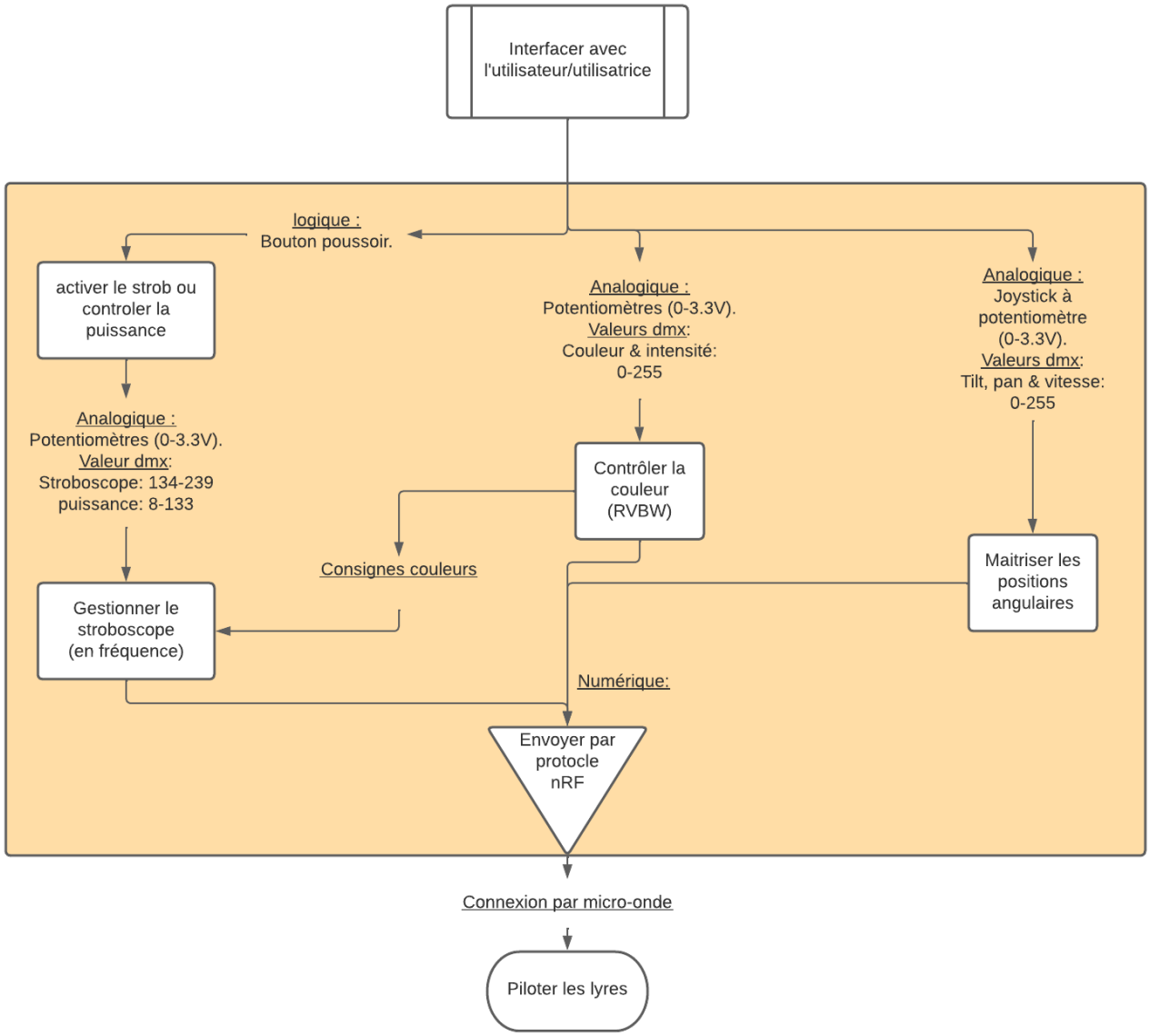
La première étape du projet fut de créer le cahier des charges de la télécommande afin d'apporter un cadre au travail à réaliser lors des séances. Après une réunion avec les membres du Laser-Wave présents pour le projet, nous sommes venus à la conclusion que nous voulions donc pouvoir contrôler plusieurs paramètres :

- Les couleurs émises par la lyre et leur intensité.
- La position et la vitesse de la lyre.
- Pouvoir utiliser l'effet stroboscope et en contrôler la fréquence.
- Pouvoir contrôler plusieurs lyres en série.

L'idée étant de pouvoir contrôler les différentes valeurs possible pour la fréquence du stroboscope, d'angles et de couleurs par des potentiomètre, afin d'avoir un contrôle le plus continu possible.
Le stroboscope doit pouvoir être activé et désactive à volonté, pour cela nous utilisons donc un bouton poussoir.
La dernière fonctionnalité que nous voulions pour ce projet est de pouvoir choisir quelle lyre nous contrôlons si celles-ci sont en série. Pour le prototype nous avons choisi d'utiliser deux lyres. Nous

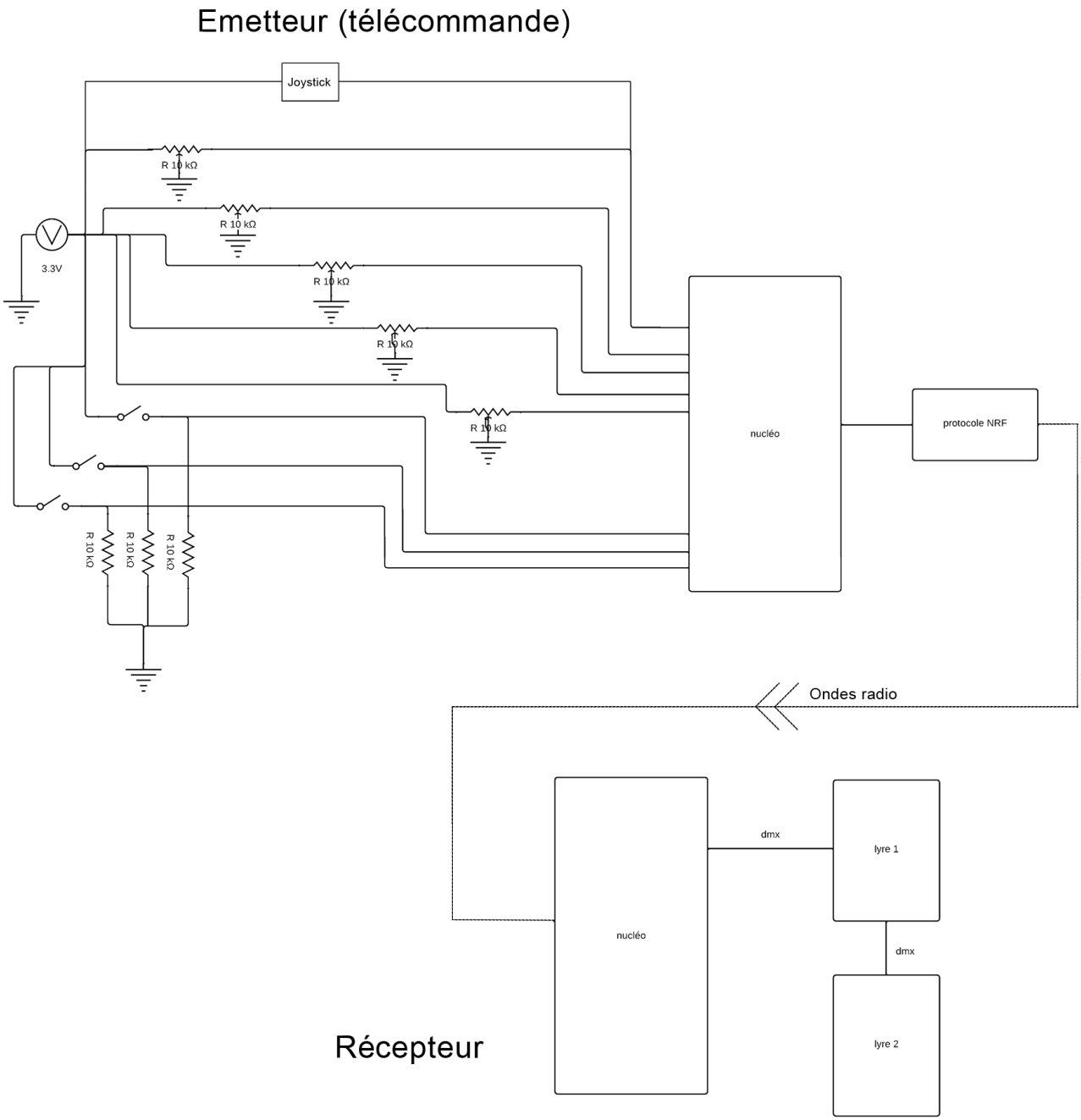
choisirons donc quelle lyre utiliser grâce à deux boutons supplémentaires.
Il faut savoir que pour envoyer des informations sur la lyre, on utilise le DMX avec une lyre en huit canaux, c'est à dire qu'elle peut recevoir huit canaux d'information ayant des valeurs comprises entre 0 et 255. chaque canal gère un des paramètre de la lyre.

Ce qui nous donne un schéma de principe comme ceci :



On remarquera que sur cette télécommande, un même potentiomètre contrôle deux fonctions dans deux configurations différentes : lorsque le stroboscope est désactivé, l'on contrôle l'intensité produite par la lyre, et dans le cas où le stroboscope est activé, l'on contrôle sa fréquence. On comprend donc qu'il n'est pas simple de contrôler à la fois la fréquence du stroboscope ainsi que l'intensité dans ce cas. Ce choix a été fait pour minimiser la taille de la télécommande, le nombre de câble et le nombre de potentiomètre, puisque l'on en a déjà utilisé 5, sans compter le joystick.

1.2 Schéma du câblage :



On peut découper ce schéma en plusieurs partie, tout d'abord ce qui est à gauche des cartes nucléo constitue l'interface homme-machine, on retrouve tout en haut le joystick, constitué de deux potentiomètres qui permet de contrôler les mouvements et vitesse des lyres, sous celui-ci ce sont 5 potentiomètres contrôlant les couleurs : rouge, vert, bleu, blanc et le stroboscope/intensité. Et en dessous de ceux-ci on retrouve les trois boutons poussoirs, deux servant à sélectionner les lyres, le troisième activant et désactivant l'effet stroboscope. Les informations saisies par l'utilisateur sont reçu par la première nucléo, qui à l'aide du protocole NRF envoie ces informations à la seconde carte nucléo qui les traite et les retransmet à chacune des deux lyres.
Plus précisément, lors du transfert, les données sont envoyées sous forme d'un tableau d'entier de longueur 10, entiers variant de 0 à 255 pour les 8 premières cases qui contiennent les informations sur la configuration qu'une lyre doit adopter. Les deux dernières cases contiennent quant à elles les entiers 0 ou 1, définissant si telle ou telle lyre est commandée.

1.3 Algorithmes :

Pour répondre à la problématique qui est de réaliser une télécommande pour lyre, nous avons dans un premier réalisé une télécommande filaire, nous n'avions donc besoin que d'un programme, et d'une carte nucléo, ce programme envoyait directement les informations cibles dans les paramètres de la lyre. Dans un second temps, l'ajout d'une liaison sans fil impliquait également l'ajout d'un second programme, et d'une seconde carte. Le premier programme permettant l'envoi de données, le second la réception, on peut voir ces deux programmes comme la séparation du programme initial en deux parties (réception des commandes saisies et application sur la lyre) avec en plus l'ajout de la liaison sans fil. Pour plus de précision on pourra se référer à l'annexe, dans lequel se trouve les deux algorithmes, le premier pour le donneur (pages 4 à 10), et le second pour le récepteur (pages 11 à 13).

2 Déroulement :

Lors de chaque séance nous avons essayé de nous répartir équitablement les tâches à accomplir, en les changeant si besoin est, pour que chacun est toujours une occupation. De cette façon nous avons plus ou moins tous participé à la conception de chaque partie de la télécommande. Cette répartition des tâches nous permettait d'avancer sans que plusieurs modifient deux zones du programme différentes en même temps, et que l'on perte du temps à corriger des erreurs évitables. Les tâches ont donc été réparties lors des séances comme décrites ci-dessous :

séance 1 :

Swen et Camille : Travail sur l'identification des besoins, sur le cahier des charges à avoir et sur l'explication du dmx et des canaux à l'équipe.

Cyril et Corentin : Début de la prise en main des lyres, le DMX et réalisation des premiers tests concluants afin de contrôler la lyre avec une carte Nucléo et des potentiomètres (avec un câble DMX entre le contrôleur et la lyre).

séance 2 :

cyril et Camille : Rédaction et mise en page du cahier des charges et du diagramme d'utilisation .
Swen et Corentin : Premier prototype de contrôleur filaire et création du premier code pour la carte Nucléo.

séance 3 :

Camille : Commence a se renseigner sur le protocole NRF afin d'implémenter le wifi dans le projet.

Cyril, Swen et Corentin : Réalisation du prototype fonctionnel de contrôleur et réalisation d'un code permettant de gérer la couleur et la position de la lyre.

A ce stade nous pouvions contrôler certains paramètre de la lyre (position et couleurs) à l'aide de potentiomètre circulaire. Le tout étant monté sur une plaquette de prototypage.

séance 4 :

Swen et Corentin : Premiers essais pour le contrôle à distance de la lyre et débogage du système.

Cyril et Camille : Commencent à regarder comment réaliser la boite du contrôleur avec une découpeuse laser et debogage du code.

séance 5 :

Cyril : Implémentation de l'effet de stroboscope dans le code.

Swen et Corentin : Debogage, branchement et programmation pour le contrôle à distance.

Camille : Réalisation des premiers jets de la boite avec la découpeuse laser.

séance 6 :

Cyril, Swen et Corentin : Débogage de la liaison sans fil.

Camille : Réalisation et finition de la boite de la télécommande (impression finale).

A ce stade nous pouvions utilisions donc deux cartes nucléo ainsi que des cartes NRF pour assurer la liaison sans fil entre les deux parties. Il ne reste alors plus qu'à installer l'ensemble de la partie réceptrice au sein de la boîte.

séance 7 :

tout le monde : fin des branchements et de la programmation. Installation de la nucléo au sein de la télécommande et tests finaux concluants.

3 Remarque sur les difficultés rencontrées :

La plupart des problèmes auxquels nous avons dû faire face sont dus à notre négligence vis à vis de certains paramètres : nomenclature dans les programmes, câblage flou ou mal fait, ou encore à la méconnaissance du fonctionnement des cartes nucléo. Cependant nous gardons tout de même en mémoire une grande source de frustration qui est la mise en place de la communication sans fil entre la télécommande et la lyre. Nous avons passé beaucoup de temps sur cette problématique (plus d'une séance), pour au final se rendre compte qu'il était nécessaire de débrancher et rebrancher le câble de la nucléo pour que les programmes fonctionnent, solution que l'on a découvert purement par hasard. Dans une moindre mesure on peut également parler de la fréquence des tikers utilisés, qui nécessitaient de ne pas être trop grande ni trop faible pour garantir un mouvement fluide de la lyre. Au delà de cela, nous n'avons pas fait face à des difficultés insurmontable, et comme on peut le déduire de la partie précédente, nous avons pu obtenir un premier prototype fonctionnel et suffisamment agréable à l'utilisation.

4 Conclusion :

Ainsi, au bout des 7 séances qui nous étaient allouées nous avons pu réaliser un prototype fonctionnel pouvant contrôler pleinement deux lyres en série. Le prototype est utilisable par le LaserWave et possède toutes les fonctionnalités citées au sein du cahier des charges. On peut tout de même faire remarquer que quelques ajouts auraient pu être fait pour améliorer l'ergonomie du contrôleur, comme d'ajouter des leds pour indiquer quelle lyre est active, mais cela n'a pas pu se faire par manque de temps.

5 Annexe

5.1 Nucléo, donneur :

```
#include "mbed.h"
#include "platform/mbed_thread.h"


#define SAMPLES      512
#define Nmoyenne     20

#include "MOD24_NRF.h"
#include "nRF24.h"

#define UDslow       1
#define UDfast       3
#define LMslow       1
#define LMfast       3

Serial       debug_pc(USBTX, USBRX);

Serial       dmx(A0, A1);
DigitalOut   out_tx(D5);
DigitalOut   start(D4);      //envoie des donnees
DigitalOut   enableDMX(D6);
AnalogIn     CV_volume(PC_1);
AnalogIn     CV_pitch(PB_0);

InterruptIn  bp_int(USER_BUTTON);


AnalogIn    LR(A3);        //left right      //variation de position
AnalogIn    UD(A2);        //up down

//DigitalIn mon_bp(A4);
AnalogIn     variationStrobo(PC_5);

//InterruptIn StroboMode(PC_4);
//DigitalIn StroboMode(PC_4);

AnalogIn    Red(PA_5);
AnalogIn    Green(PA_6);
AnalogIn    Blue(PA_7);
AnalogIn    White(PB_1);

Ticker POSITION;
Ticker COULEUR;
Ticker BOUTON;

DigitalIn my_bp(PC_4);
DigitalIn bp_Lyre1(PC_2);
DigitalIn bp_Lyre2(PC_3);
```

```
char dmx_data[SAMPLES] = {0};

int x;

int old_bp_str, new_bp_str;
int old_bp_1, new_bp_1;
int old_bp_2, new_bp_2;

char Rm[Nmoyenne+1]= {0};
char Gm[Nmoyenne+1]= {0};
char Bm[Nmoyenne+1]= {0};
char Wm[Nmoyenne+1]= {0};

char nb = 0;

void initDMX();
void updateDMX();

void colorset();
void positionset();
void stroboset();
void Lyre1();
void Lyre2();
void stroboONOFF();
void strobo(double old_bp,double new_bp);
int moyenne(char tab[]);
void boutonset();

bool BOOLLyre1;
bool BOOLLyre2;
bool BOOLstrobo;

int main()
{
    debug_pc.baud(9600);
    debug_pc.printf("Essai DMX512\r\n");
    initNRF24();



    POSITION.attach(&positionset,0.035);
    COULEUR.attach(&colorset,0.01/Nmoyenne);
    BOUTON.attach(&boutonset,0.1);


    dataToSend[0] = 0;
    dataToSend[1] = 0;
    dataToSend[2] = 0;
    dataToSend[3] = 0;
    dataToSend[4] = 0;
    dataToSend[5] = 0;
    dataToSend[6] = 0;
```

```
    dataToSend[7] = 0;
    dataToSend[8] = 0;
    dataToSend[9] = 0;

    while(1) {
        testNRF24();
        wait_us(5000);
    }
}

// Fonction utilisee pour appliquer le m me tiker pour les 3 boutons poussoir
void boutonset()
{
    Lyre1();

    Lyre2();

    stroboONOFF();

}


// Activation/Desactivation de la lyre 1
void Lyre1()
{
    new_bp_1 = bp_Lyre1;
    if((old_bp_1 != new_bp_1) && (new_bp_1 == 1)) {    // front montant
        if (BOOLLyre1==1) {
            BOOLLyre1 = 0;
        } else {
            BOOLLyre1=1;
        }
    }
    old_bp_1 = new_bp_1;
}

// Activation/Desactivation de la lyre 2
void Lyre2()
{
    new_bp_2 = bp_Lyre2;
    if((old_bp_2 != new_bp_2) && (new_bp_2 == 1)) {    // front montant
        if (BOOLLyre2==1) {
            BOOLLyre2 = 0;
        } else {
            BOOLLyre2=1;
        }
    }
    old_bp_2 = new_bp_2;
}

// Activation/Desactivation de l'effet stroboscope
void stroboONOFF()
{
```

```
    new_bp_str = my_bp;
    if((old_bp_str != new_bp_str) && (new_bp_str == 1)) {    // front montant
        if (BOOLstrobo==1) {
            BOOLstrobo = 0;
        } else {
            BOOLstrobo=1;
        }
    }
    old_bp_str = new_bp_str;
}

//Commande en position et vitesse de la lyre
void positionset()
{

    if ((LR.read() <=0.125) && (dataToSend[0] > 2)) {
        dataToSend[0]-=LMfast;
        dataToSend[7] = 0;
    } else if (LR.read() <=0.475 && dataToSend[0] > 0) {
        dataToSend[0]-=LMslow;
        dataToSend[7] = 100;
    }

    else if (LR.read() <=0.625)
        dataToSend[0]+=0;

    else if (LR.read() <=0.875 && dataToSend[0] < 255) {
        dataToSend[0]+=LMslow;
        dataToSend[7] = 100;
    }

    else if (LR.read() <=2 && dataToSend[0]+LMfast < 255) {
        dataToSend[0]+=LMfast;
        dataToSend[7] = 0;
    }

    if (UD.read() <=0.125 && dataToSend[1]-UDfast > 0) {
        dataToSend[1]-=UDfast;
        dataToSend[7] = 0;
    }

    else if (UD.read() <=0.475 && dataToSend[1] > 0) {
        dataToSend[1]-=UDslow;
        dataToSend[7] = 100;
    } else if (UD.read() <=0.625)
        dataToSend[1]+=0;

    else if (UD.read() <=0.875 && dataToSend[1]+UDslow < 255) {
        dataToSend[1]+=UDslow;
        dataToSend[7] = 100;
    }

    else if (LR.read() <=1 && dataToSend[1] < 253) {
```

```
        dataToSend[1]+=UDfast;
        dataToSend[7] = 0;
    }
}

//renvoi la moyenne d'un tableau d'entier
int moyenne(char tab[])
{
    int i;
    int s =0;
    for (i=0; i<tab[0]; i++)
        s+=tab[i+1];
    return s/tab[0] ;
}

// Fonction contr lant l'utilisation des deux lyres, et moyennant les valeurs
void colorset()
{

    if (BOOLLyre1 == true) {
        dataToSend[8] = 1;
    } else {
        dataToSend[8]= 0;
    }

    if (BOOLLyre2 == true) {
        dataToSend[9] = 1;
    } else {
        dataToSend[9]= 0;
    }

    if (Rm[0]< Nmoyenne) {
        Rm[0]+=1;
        Rm[Rm[0]]=Red.read() * 255;
    } else {
        dataToSend[3] = moyenne(Rm);
        Rm[0]=0;
    }

    if (Gm[0]< Nmoyenne) {
        Gm[0]+=1;
        Gm[Gm[0]]=Green.read() * 255;
    } else {
        dataToSend[4] = moyenne(Gm);
        Gm[0]=0;
    }

    if (Bm[0]< Nmoyenne) {
        Bm[0]+=1;
        Bm[Bm[0]]=Blue.read() * 255;
    } else {
        dataToSend[5] = moyenne(Bm);
        Bm[0]=0;
```

```
    }

    if (Wm[0]< Nmoyenne) {
        Wm[0]+=1;
        Wm[Wm[0]]=White.read() * 255;
    } else {
        dataToSend[6] = moyenne(Wm);
        Wm[0]=0;
    }

    if (BOOLstrobo == true) {
        dataToSend[2] = 104*variationStrobo.read() + 135;
    } else {
        dataToSend[2]=134*variationStrobo.read();
    }
}

// Initialisation du DMX
void initDMX()
{
    dmx.baud(250000);
    dmx.format (8, SerialBase::None, 2);
    enableDMX = 0;
    for(int k = 0; k < SAMPLES; k++) {
        dmx_data[k] = 0;
    }
    updateDMX();
}

// Mis    jour du DMX
void updateDMX()
{
    enableDMX = 1;
    start = 1;
    out_tx = 0;
    wait_us(88);
    out_tx = 1;
    wait_us(8);
    out_tx = 0;
    start = 0;
    dmx.putc(0);
    for(int i = 0; i < SAMPLES; i++) {
        dmx.putc(dmx_data[i]);
    }
}
```

5.2 Protocole NRF24, donneur :

```
#include "nRF24.h"

#define TRANSFER_SIZE    10

nRF24L01P        nRF24_mod(PC_12, PC_11, PC_10, PA_15, PB_7, PA_13);

// MOSI, MISO, SCK, CSN, CE, IRQ

char k;
int i;
char     dataToSend[TRANSFER_SIZE] = {0};
char     dataReceived[TRANSFER_SIZE] = {0};

char         rxDataCnt;


// Fonction d'initialisation du module BT nRF24L01
void initNRF24(){
    nRF24_mod.powerUp();
    wait_us(100000);
    nRF24_mod.setAirDataRate(NRF24L01P_DATARATE_250_KBPS);
    nRF24_mod.setRfFrequency(2400);
    wait_us(100000);
    debug_pc.printf( "nRF24L01+ Frequency     : %d MHz\r\n",  nRF24_mod.getRfFr
    debug_pc.printf( "nRF24L01+ Output power  : %d dBm\r\n",  nRF24_mod.getRfOu
    debug_pc.printf( "nRF24L01+ Data Rate     : %d kbps\r\n", nRF24_mod.getAirD
    debug_pc.printf( "Transfers are grouped into %d characters\r\n", TRANSFER_
    nRF24_mod.setTransferSize( TRANSFER_SIZE );
    nRF24_mod.setReceiveMode();
    nRF24_mod.enable();
}

// Fonction de test du module BT nRF24L01
void testNRF24(void){

        nRF24_mod.write( NRF24L01P_PIPE_P0, dataToSend, TRANSFER_SIZE );
        for ( int i = 0; i < 10; i++ ) {
            debug_pc.printf(" %d \t", dataToSend[i]);
        }
        debug_pc.printf( "SENDED\r\n");
        wait_us(10000);

}
```

5.3 Nucleo, receveur :

```
#include "mbed.h"
#include "platform/mbed_thread.h"


#define SAMPLES      512
char dmx_data[SAMPLES] = {0};

int i;


#include "MOD24_NRF.h"
#include "nRF24.h"


void initDMX();
void updateDMX();


#define Nmoyenne     20

#define UDslow       1
#define UDfast       3
#define LMslow       1
#define LMfast       3

Serial       debug_pc(USBTX, USBRX);

Serial       dmx(A0, A1);
DigitalOut   out_tx(D5);
DigitalOut   start(D4);
DigitalOut   enableDMX(D6);
AnalogIn     CV_volume(PC_1);
AnalogIn     CV_pitch(PB_0);

InterruptIn  bp_int(USER_BUTTON);


AnalogIn    LR(A3);        //left right      //variation de position
AnalogIn    UD(A2);        //up down

AnalogIn    Red(PA_5);
AnalogIn    Green(PA_6);
AnalogIn    Blue(PA_7);
AnalogIn    White(PB_1);

Ticker POSITION;
Ticker COULEUR;
Ticker STROBO;
```

```
int main() {
    debug_pc.baud(9600);
    debug_pc.printf("Essai DMX512\r\n");


    initDMX();
    wait(0.5);
    initNRF24();



        dmx_data[0] = 0;
        dmx_data[1] = 0;
        dmx_data[2] = 0;
        dmx_data[3] = 0;
        dmx_data[4] = 0;
        dmx_data[5] = 0;
        dmx_data[6] = 0;
        dmx_data[7] = 0;
        dmx_data[8] = 0;
        dmx_data[9] = 0;
    while(1) {

        if ( nRF24_mod.readable() ) {
        testNRF24();

        if (dataReceived[8]==1){
            for (i=0;i<8;i++){
            dmx_data[i]=dataReceived[i];
            }
        }

        if (dataReceived[9]==1){
            for (i=0;i<8;i++){
            dmx_data[8+i]=dataReceived[i];
            }
        }

       }

        updateDMX();

    }
}


void initDMX(){
    dmx.baud(250000);
    dmx.format (8, SerialBase::None, 2);
    enableDMX = 0;
    for(int k = 0; k < SAMPLES; k++){
        dmx_data[k] = 0;
    }
```

```
    updateDMX();
}

void updateDMX(){
        enableDMX = 1;
        start = 1;
        out_tx = 0;
        wait_us(88);
        out_tx = 1;
        wait_us(8);
        out_tx = 0;
        start = 0;
        dmx.putc(0);
        for(int i = 0; i < SAMPLES; i++){
            dmx.putc(dmx_data[i]);
        }
}
```

5.4 Protocole NRF24, receveur :

```
#include "nRF24.h"

#define TRANSFER_SIZE    10

nRF24L01P        nRF24_mod(D11, D12, D13, D10, D9, A5);
// MOSI, MISO, SCK, CSN, CE, IRQ

char k;

char     dataReceived[TRANSFER_SIZE] = {0};

char         rxDataCnt;


// Fonction d'initialisation du module BT nRF24L01
void initNRF24(){
    nRF24_mod.powerUp();
    wait_us(100000);
    nRF24_mod.setAirDataRate(NRF24L01P_DATARATE_250_KBPS);
    nRF24_mod.setRfFrequency(2400);
    wait_us(100000);
    debug_pc.printf( "nRF24L01+ Frequency    : %d MHz\r\n",  nRF24_mod.getRfFr
    debug_pc.printf( "nRF24L01+ Output power : %d dBm\r\n",  nRF24_mod.getRfOu
    debug_pc.printf( "nRF24L01+ Data Rate    : %d kbps\r\n", nRF24_mod.getAirD
    debug_pc.printf( "Transfers are grouped into %d characters\r\n", TRANSFER_
    nRF24_mod.setTransferSize( TRANSFER_SIZE );
    nRF24_mod.setReceiveMode();
    nRF24_mod.enable();
}

// Fonction de test du module BT nRF24L01
void testNRF24(){
```

```
    /* Lecture donnee depuis nRF24 */
        nRF24_mod.read( NRF24L01P_PIPE_P0, dataReceived, TRANSFER_SIZE);
        rxDataCnt = nRF24_mod.read( NRF24L01P_PIPE_P0, dataReceived, TRANSFER_
        debug_pc.printf("\n");

        for ( int i = 0; i < TRANSFER_SIZE; i++ ) {
            debug_pc.printf("%d\t", dataReceived[i]);
        }
}
```