PROTIS : livrable de fin de projet

Nathanaell Benon - Fabien Daheron
Baptiste Leprovost - Aymane Rouabah

6 avril 2023

Table des matières

1 Introduction

Le présent rapport concerne le projet PROTIS 2022_L058. Ce projet consiste en la réalisation d'une interface permettant de piloter des projecteurs (pilotés par DMX) grâce à un contrôleur MIDI, une AKAI dans notre cas.

Les attendus sont les suivants :
— Piloter jusqu'à 16 projecteurs séparément
— Proposer 3 modes
— Permettre de changer la configuration sans devoir recompiler
Les trois modes demandés sont les suivants :

Mode 1 : Permet à l'utilisateur de piloter chaque projecteur indépendamment dans le but de piloter leur intensité lumineuse, leur couleur et leur orientation. Le pilotage se fera en temps direct de façon la plus facile d'utilisation possible.

Mode 2 : Permet à l'utilisateur de piloter des ensembles de projecteurs en simultané. Le pilotage se fera en temps direct de façon la plus facile d'utilisation possible.

Mode 3 : Permet à l'utilisateur de produire une séquence lumineuse pré-enregistrée sans devoir la jouer manuellement de nouveau.

1.1 Schéma fonctionnel

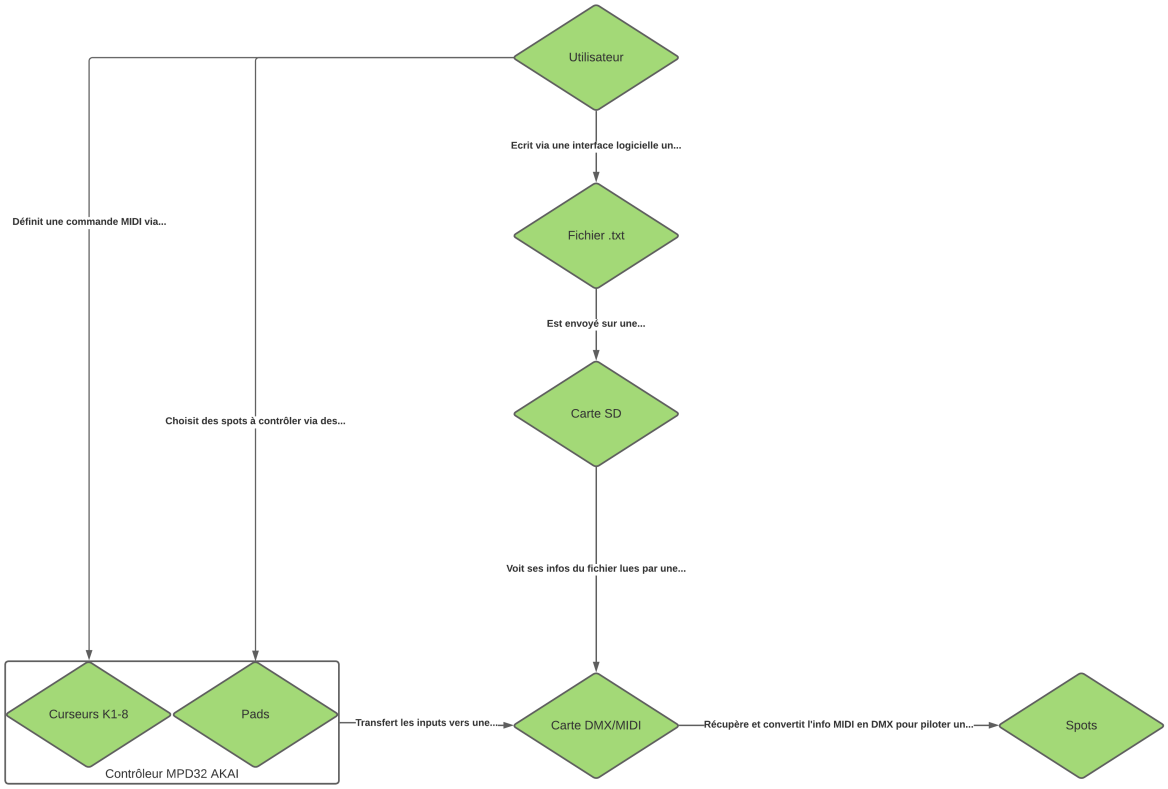

FIGURE 1.1 – Schéma fonctionnel

2 Fonctions élémentaires utiles au projet

Dans cette partie, nous allons vous présenter les fonctions élémentaires que nous avons utilisé dans notre projet, et expliquer leur fonctionnement. Les étapes, dans l'ordre où nous les avons réalisées avant de les réunir ensemble, sont les suivantes :

— Détecter et identifier les octets MIDI reçus
— Mettre à jour les canaux DMX pour piloter les lumières
— Créer la liste des canaux associés aux octets en lisant une carte SD

Ces étapes permettent de créer les programmes complets de chaque mode, qui seront présentés en détail dans les parties suivantes.

Pour les deux premières étapes, nous avons réutilisé les fonctions écrites dans un précédent projet, disponible à cette adresse.

2.1 Première partie : MIDI

La première étape a été de vérifier que nous étions bien capable de récupérer et d'identifier les octets MIDI reçus. Pour cela, nous utilisons simplement deux fonctions de la librairie MIDI créée dans ce projet, il s'agit de `isNoteMIDIdetected` et de `isCCMIDIdetected`.

Ces fonctions exploite en réalité une interruption détectant l'arrivée d'un octet MIDI, qui déclenche la fonction `ISR_midi_in` :

```
void ISR_midi_in(void){
    debug_out = !debug_out;
    char data = midi.getc();
    if(data >= 128)
        cpt_midi = 0;
    else
        cpt_midi++;
    midi_data[cpt_midi] = data;
    if(cpt_midi == 2){
        cpt_midi = 0;
        if((midi_data[0] == MIDI_NOTE_ON) || (midi_data[0] == MIDI_NOTE_OFF)){
            new_note_midi = 1;
            note_data = midi_data[1];
            velocity_data = midi_data[2];
        }
        else{
            if(midi_data[0] == MIDI_CC){
                new_data_midi = 1;
                control_ch = midi_data[1];
                control_value = midi_data[2];
            }
        }
    }
}
```

Concrètement, cette fonction affecte les variables note_data et velocity_data si l'octet détecté est celui d'un pad, et les variables control_ch et control_value si l'octet est celui d'un curseur. Dans les deux cas, la première variable correspond à l'identifiant du pad ou du curseur, et la deuxième à sa valeur (la pression dans le cas du pad, et la position dans le cas du curseur). Ce sont ces variables que nous utilisons.

Dans le programme, il suffit donc d'utiliser les fonctions `isNoteMIDIdetected` et `isCCMIDIdetected`, qui se contentent de vérifier les valeurs de `new\_note\_midi` ou `new\_data\_midi`, afin de savoir si un pad ou un curseur a été sollicité.

En réalité, les seules modifications faites à cette librairie de notre côté sont le changement des ports, puisque nous n'utilisons pas la même carte (F767ZI et non L476RG).

2.2 Deuxième partie : DMX

Ici aussi, nous utilisons principalement les librairies précédentes, qui nous permettront de commander les groupes de lumière via DMX. La fonction exploitée est simplement `updateDMX`.

```
void updateDMX(){
    enableDMX = 1;
    start = 1;        // /start
    out_tx = 0;       // break
    wait_us(88);
    out_tx = 1;       // mb
    wait_us(8);
    out_tx = 0;       // break
    start = 0;
    dmx.putc(0);      // Start
    for(int i = 0; i < SAMPLES; i++){ // SAMPLES = nombre de canaux, 512 ici
        dmx.putc(dmx_data[i]);      // data
    }
    wait_us(23000); // time between frame
}
```

Cette fonction, un peu particulière notamment au vu des temps d'attentes, est conçue pour former le train DMX avec sa forme spécifique, aussi nous n'y avons pas touché. Une fois encore, la seule différence ici aura été de changer les ports.

Pour commander un groupe de projecteur, il suffit de changer les valeurs de la liste `dmx\_data` selon le canal qu'écoute le projecteur, puis d'appeler la fonction `updateDMX`. Nous détaillerons cela plus tard, lorsque de la présentation du programme complet.

2.3 Troisième partie : lire un fichier sur la carte SD et initialiser le programme

Précédemment, nous avons vu que nous étions capables de récupérer les octets MIDI mais également de commander les canaux DMX. Il ne reste donc qu'à associer les deux, et ce via un fichier écrit sur une carte SD plutôt qu'écrit en dur dans le programme, pour ne pas avoir à recompiler.

Pour cela, nous avons décidé d'utiliser une structure particulière de fichier de configuration, dont voici un exemple (les commentaires ne doivent pas figurer dans le fichier !) :

```
2 // Nombre de pad que nous souhaitons utiliser
0x43 // Octet associe au premier pad que nous souhaitons utiliser
16 // Canal associe
0x12
82
```

Les octets dépendent de la configuration du clavier, il faut donc vérifier quel octet est envoyé par la console MIDI lorsque le pad choisi est pressé.

```
void initialisation_octets(){
    // Montage de la carte SD
    int err = fileSystem.mount(&blockDevice);
    printf("%s\n", (err ? "Fail, montage impossible :(" : "OK"));
    if (!err) {
        // On essaye d'ouvrir le fichier de config
        FILE*    f = fopen("/fs/config1.txt", "r+");
        printf("%s\n", (!f ? "Fail, fichier introuvable :(" : "OK"));
        if (f) {
            fscanf(f, "%d", &nombre_pad); // Cette fonction lit les lignes du fichier
    , une a la fois. Premier appel -> premiere ligne
            printf("Nombre pad : %d\r\n",nombre_pad);
            octets = (unsigned char*) malloc(nombre_pad*sizeof(unsigned char));
```

```
13              offset_dmx = (int*) malloc(nombre_pad*sizeof(int));
14              // Go through and increment the numbers
15              for (int i = 0; i < nombre_pad; i++) {
16
17                  fscanf(f, "%x", &octets[i]);
18                  fscanf(f, "%d", &offset_dmx[i]);
19                  printf("Verif Octet %d : %x . Adresse DMX : %d !\r\n", i, octets[i],
    offset_dmx[i]); // pour controle visuel de la bonne lecture du fichier
20              }
21          }
22          // Close the file
23          err = fclose(f);
24      }
25      // Demontage de la carte SD...
26      err = fileSystem.unmount();
27      printf("%s\n", (err < 0 ? "Fail, rate le demontage :(" : "OK"));
28  }
```

Cette fonction va donc monter la carte SD, ouvrir le fichier prénommé `config1.txt` dont nous avons fourni un exemple plus haut, puis créer les listes `octets` et `offset_dmx` qui contiennent respectivement quel octet est associé à quelle adresse du canal DMX souhaité.

3 Programme complet du mode 1

3.1 Bilan du mode 1

Le programme étant un peu long (environ 200 lignes), nous ne le retranscrirons pas en texte ici. Le code source est largement commenté. Nous en décrivons toutefois le principe détaillé ici.

Ce programme comporte, outre les librairies MIDI et DMX, une fonction `initialisation_octets` présentée ci-dessus ainsi que la fonction `main`. Le principe est le suivant : le programme initialise la liste comme décrit dans la partie précédente, puis attend la détection d'octets MIDI. S'il s'agit de la pression sur un pad, alors le programme identifie à quel canal DMX cela correspond, canal qui sera identifié via la variable `offset_dmx`. S'il s'agit du déplacement d'un curseur, alors le programme identifie quel curseur est déplacé, et quel sera le canal à modifier en conséquence.

En effet, si notre groupe de projecteur est réglé sur le canal 45 par exemple, alors une modification du canal 45 modifiera le premier paramètre des projecteurs (typiquement l'intensité globale, selon les projecteurs), le canal 46 modifiera le deuxième paramètre et ainsi de suite. Ainsi, si le premier curseur est déplacé, on modifie le canal `offset_dmx`, si c'est le deuxième curseur on modifie le canal `offset_dmx+1` etc...
En parallèle, le programme met régulièrement à jour les canaux DMX avec la liste que nous avons mis à jour `dmx_data`, ce qui provoque la mise à jour de l'état des projecteurs.

3.2 Lien de la documentation d'utilisation (version pour le prototype)

Pour le pilotage du prototype du mode 1, une documentation sous forme de notice d'utilisation a été réalisée pour permettre aux différents membres du groupe et à l'équipe PROTIS d'utiliser le système. Vous trouverez ce document au lien suivant.

Doc_utilisation.pdf

4 Programme complet du mode 2

Le second mode demandé dans le cahier des charges est la possibilité de piloter plusieurs projecteurs en même temps. Le mode 1 permet de le faire en mettant les projecteurs que l'on souhaite piloter simultanément sur la même adresse de lecture du canal DMX. Cependant, la sélection de l'adresse ce fait directement sur le projecteur ce qui oblige de paramétrer les projecteurs. Dans le cadre d'une scène où les

projecteurs seraient installés au plafond, modifier l'adresse de lecture sur le canal DMX d'un projecteur peut être fastidieux. De ce fait on souhaite reprendre le même procédé que pour le mode 1 mais où un appui sur un pad contrôle plusieurs canaux DMX. Le fonctionnement de pilotage avec les curseurs sera strictement identique.

4.1 Modification du fichier de configuration sur la carte SD

Pour permettre au programme sur la carte nucleo d'associer à un pad les canaux DMX associés, nous allons commencer par modifier la structure du document texte de configuration sur la carte SD. Dans un premier temps l'utilisateur renseigne le nombre de PAD qu'il souhaite utiliser et le nombre de projecteurs (soit de canaux DMX) qu'il souhaite utiliser. La seule subtilité est que si un projecteur (ie un canal DMX) est associé à plusieurs pads alors il devra être compté plusieurs fois. Ensuite pour chaque octet associé à un pad, l'utilisateur renseigne le nombre des canaux DMX associés puis le numéro correspondant à ces canaux. Un exemple de nouveau fichier de configuration est alors (sans les commentaires) :

```
3 // Nombre de pad que nous souhaitons utiliser
5 // Nombre de canaux qui seront associés (on compte les doublons)
0x43 // Octet associe au premier pad que nous souhaitons utiliser
2 // Nombre de canaux associés à ce pad
16 // Premier canal associé à ce pad
59 // Second canal
0x12
2
82
16
0x13
1
115
// On notera que 5 = 2 + 2 + 1
```

4.2 Modification du cœur du programme

Dans le mode 1 présenté précédemment, lorsqu'un utilisateur appuie sur un pad alors à l'aide de `note_data` nous pouvions connaître quel est le canal associé. Nous retenons ce canal en mémoire pour que dès lors qu'un curseur est déplacé nous puissions correctement modifier `dmx_data` selon les commandes de l'utilisateur.

Dans ce mode 2, il faut garder en mémoire l'octet du dernier pad pressé. Lorsque l'utilisateur déplace un curseur, le programme cherchera les canaux différents canaux associés et modifiera correctement `dmx_data`. Pour cela, la structure des listes `offset_dmx` et `octets` est modifiée dans cette partie du mode 2. Pour éviter les confusions nous les appellerons `offset_dmx2` et `octets2`. la structure étant différente, la fonction `initialisation_octets` l'est aussi.

```
void initialisation_octets(){
    //Montage de la carte SD
    int err = fileSystem.mount(&blockDevice);
    printf("%s\n", (err ? "Fail, montage impossible :(" : "OK"));
    if (!err) {
        // On essaye d'ouvrir le fichier de config
        FILE*   f = fopen("/fs/configM.txt", "r+");
        printf("%s\n", (!f ? "Fail, fichier introuvable :(" : "OK1"));
        if (f) {
            fscanf(f, "%d", &nombre_pad);
            fscanf(f, "%d", &nombre_dmx);
            printf("Nombre pas : %d\r\n",nombre_pad);
            printf("Nombre dmx : %d\r\n",nombre_dmx);
            // Création des tableaux a la bonne taille de octetc2 et offset_dmx2
            octets2 = (unsigned char*) malloc(nombre_dmx*sizeof(unsigned char));
            offset_dmx2 = (int*) malloc(nombre_dmx*sizeof(int));
```

```
            int nombre_adresse_du_pad;
            int position_de_remplisage = 0;
            // Go through and increment the numbers
            for (int i = 0; i < nombre_pad; i++) {
                // On lit l'octet et on le sauvegarde dans dernier octets
                fscanf(f, "%x", &dernier_octets);
                // Pour le nombre de canaux associe a ce pad on recupere leur valeur
                fscanf(f, "%d", &nombre_adresse_du_pad);
                for (int j = 0; j < nombre_adresse_du_pad; j++) {
                    octets2[position_de_remplisage] = dernier_octets;
                    fscanf(f, "%d", &offset_dmx2[position_de_remplisage]);
                    position_de_remplisage++;
                }
            }
        }
        //Verification du remplissage
        printf("Octet 1 : %x ; Octet 2 : %x ; Octet 3 : %x \r\n",octets2[0],octets2
    [1], octets2[2]);
        printf("Adresse 1 : %d ; Adresse 2 : %d ; Adresse 3 : %d \r\n",offset_dmx2
    [0],offset_dmx2[1], offset_dmx2[2]);

        // Close the file
        err = fclose(f);

    }
    // Unmounting SD...
    err = fileSystem.unmount();
    printf("%s\n", (err < 0 ? "Fail, rate le demontage :(" : "OK2"));
}
```

Pour mieux comprendre ce que fait la fonction d'initialisation on vous propose ci-dessous d'étudier la structure de `offset_dmx2` et `octets2` dans l'exemple du fichier de configuration fourni ci-dessus. On a alors les listes suivantes. Pour chaque entier i compris compris entre 0 et le nombre de canaux utilisés - 1, le canal `offset_dmx2[i]` est associé au pad d'octet `octets2[i]`.

```
octets2 = [ 0x43 , 0x43 , 0x12 , 0x12 , 0x13];
offset_dmx2 = [ 16 , 59 , 82 , 16 , 115];
```

Ainsi, lorsque l'utilisateur appuie sur un pad alors on récupère l'octet associé grâce à `note_data` que l'on enregistre dans une variable `dernier_octets`. Puis à chaque fois qu'un curseur est déplacé par l'utilisateur, on parcours la liste `octets2` et lorsqu'on rencontre `dernier_octets` on modifie `dmx_data` en fonction du canal `offset_dmx2` associé. On peut alors piloter plusieurs canaux DMX avec un même pad.

```

    if(isNoteMIDIdetected()){ // La pression sur un pad est détecté
            dernier_octets = note_data;
            resetNoteMIDI();
        }
    if(isCCMIDIdetected()){ // On detecte le déplacement d'un curseur Cela cree deux
    variables : control_ch octet identifiant le pad ; control_value position du
    curseur
            printf("Curseur : %x ; Position = %x \r\n",control_ch,control_value);
            if(control_ch == 1){curseur[0] = 2 * control_value;} // Si le premier
    curseur est deplace alors on affecte la nouvelle valeur au premier element du
    tableau curseur.
            if(control_ch == 2){curseur[1] = 2 * control_value;}
            if(control_ch == 3){curseur[2] = 2 * control_value;}
            //etc... Supprime uniquement sur ce document pour compacter
            if(control_ch == 11){curseur[10] = 2 * control_value;}
            resetCCMIDI();
        }//*/
        // On vient maintenant pour chaque canal associe au dernier pad presse
    modifier dmx_data selon la demande de l'utilisateur
```

```
16          for (int i = 0; i < nombre_dmx; i++){
17              if(dernier_octets == octets2[i]){
18                  X = offset_dmx2[i];
19                  dmx_data[X - 1] = curseur[0];
20                  dmx_data[X + 0] = curseur[1];
21                  dmx_data[X + 1] = curseur[2];
22                  //etc... Supprime uniquement sur ce document pour compacter
23                  dmx_data[X + 6] = curseur[7];
24                  // Ces trois prochains curseurs sont censes servir aux lyres pour le
        deplacement. Le decalage de 3 est du a notre souhait d'avoir le meme curseur d'
        intensite.
25                  dmx_data[X - 1 - 3] = curseur[8];
26                  dmx_data[X + 0 - 3] = curseur[9];
27                  dmx_data[X + 1 - 3] = curseur[10];
28              }
29          }
30          updateDMX();
31          wait_us(1);
```

4.3 Conclusion et avantages du mode 2

On pourra noté à travers l'exemple du fichier de configuration sur la carte SD donné que ce mode permet pour chaque pad d'associer plusieurs projecteurs ou lyres. De plus le nombre de projecteurs n'est pas nécessairement le même pour chaque pad et il est possible d'en mettre que 1 par pad. Ainsi, le mode 2 comprend le mode 1, c'est plus une extension du mode 1 plutôt qu'un autre mode.

Pour autant, ce résultat peut être obtenu avec le mode 1 si l'on met plusieurs projecteurs sur la mettre adresse d'écoute. mais ce que permet le même 2 par rapport au mode 1 est qu'un projecteur peut être associé à plusieurs pad et donc plusieurs groupes de projecteurs dans une même configuration.

On notera cependant, qu'il est nécessaire que les projecteurs sur un même pad soient d'une même marque sinon un curseur ne correspondra pas à la même information pour chaque projecteur.

Pour conclure, ce programme compile bien mais le démontage de la carte SD ne fonctionne pas bien que ce soit les mêmes opérations que dans le mode précédant. Le programme n'affiche même pas le message d'erreur "Fail rate le démontage" mais ce contente de ce relancer en boucle. Après 2 heures de réflexion avec votre collègue de SOLEC, monsieur VILLOU, nous n'avons pas eu le temps de comprendre pourquoi cela ne fonctionnait pas. Si l'on affiche les tableaux `offset_dmx2` et `octets2`, que l'on fait un `wait_us(1000000)` ou que l'on fait n'importe qu'elle autre action avant de démonter, tout fonctionne sauf le démontage malgré la similarité avec le mode 1.

Puisque les tableaux `offset_dmx2` et `octets2` sont correctement fabriqués donc nous avons confiance sur le fonctionnement du reste du programme puisque le fonctionnement est similaire au mode 1 sur les fonctions utilisées.

5 Programme complet du mode 3

les deux modes précèdent requièrent que l'utilisateur modifie en temps réel la couleur et la luminosité de chaque projecteur. Cependant, il peut arriver que ce dernier souhaite jouer plusieurs fois la même série de couleur. Pour cela, ce troisième mode lui permettra d'enregistrer une séquence pour la répéter. Initialement nous aurons aimé que le programme puisse enregistrer une série d'opération et l'associer à un pad ce qui aurait été très pratique. Cependant nous n'avons pas réussi à trouver comment faire fonctionner cette programmation. L'option aujourd'hui mise en place consiste à écrire sur un fichier texte la séquence d'action à effectuer. Cette séquence sera associée à l'octet de ce pad et à chaque fois que l'utilisateur appuiera sur ce pad la séquence se lancera automatiquement.

5.1 Format des fichiers de séquences

Pour chaque séquence, l'utilisateur devra écrire un fichier texte nommé `Sequence{x}.txt`. Pour l'instant le programme permet d'associer au maximum 16 séquences soit une par pad sur un même clavier. De ce fait les fichiers lisibles par le programme sont `Sequence1.txt` à `Sequence16.txt`. Les fichiers de séquences contiennent pour chaque action l'adresse du canal DMX à modifier, la nouvelle valeur à affecter et la temps d'attente avant d'effectuer l'action suivante. A la fin du document, le dernier temps d'attente sera de -1 pour indiquer au programme de finir la lecture de la séquence. Voici un exemple de fichier séquence (sans les commentaires) :

```
// Nom du fichier : Sequence9.txt
1 // On souhaite modifier le canal 1
50 // On souhaite affecter au canal 1 la valeur 50
100 //Temps d'attente en ms avant la prochaine action
15 // On souhaite modifier le canal 15
150 // On souhaite lui affecter la valeur 150
3000  // On attend 3 secondes avant la prochaine action
2
0
0
3
150
2000
2
150
-1 // Fin de la sequence sur un temps negatif de -1
```

Pour orchestrer les différentes séquences et indiquer au programme quelle séquence est associée à quel octet (ie quel pad), l'utilisateur dispose sur la carte SD un autre document texte nommé `AdresseSequence.txt` qui contient le nombre de séquences utilisées dans la configuration et le lien entre le numéro de la séquence et les octets du clavier MIDI. Voici un exemple de fichier pour enregistrer associer 2 séquences dont la séquence ci-dessus (la numéro 9) est associée au pad d'octet 0x11. Les commentaires ne doivent pas figurer sur le document texte.

```
2 // Nombre de sequences à enregistrer
0x11 // octet du PAD
9 // numero de la sequence de 1 a 16
0x12
2
```

5.2 Fonctionnement du programme

Sans trop rentrer dans les détails, comme précédemment une fonction d'initialisation permet de créer les tableaux `octets_PAD_sequences` et `numeros_PAD_sequences` avec la lecture du fichier `AdresseSequence.txt`. Ces listes comportent respectivement les octets utilisés pour les séquences et les numéros de séquence associés. Dans l'exemple ci-dessus ces tableaux seraient les suivants :

```
octets_PAD_sequences = [0x11 , 0x12];
numeros_PAD_sequences = [9 , 2];
```

Lorsque l'utilisateur appuie sur la pad correspondant à un des octets associé à une séquence, le programme récupère le numéro de séquence {x} associé et effectue la fonction suivante avec en entrée le fichier `Sequence{x}.txt` qui permet de jouer la séquence.

```
void lecture_sequence(FILE* fichier){
// affichage_texte est un booleen global qui permet d'eviter d'afficher du texte sur
    teraterm si pas necessaire pour ne pas ralentir le programme.
if(affichage_texte){printf("Lancement de la sequence \r\n");}
int XXX ;   // XXX est EGAL a l'adresse du train dmx que l'on souhaite modifier
```

```
// Attention cela correspond à la position XXX-1 dans la liste dmx_data
int AffectedValue; // AffectedValue est EGAL a la value que l'on souhaite affecter à la modification du train dmx
int Time_ms = 0; // Time_s est EGAL au temps d'attente entre chaque lecture et action en milli-secondes; A la fin du fichier Time_ms = -1 .
while(Time_ms != -1) {
    fscanf(fichier, "%d", &XXX);
    fscanf(fichier, "%d", &AffectedValue);
    fscanf(fichier, "%d", &Time_ms);
    if(affichage_texte){printf("Prochaine etape de la sequence : XXX : %d , Value : %d , Time_ms : %d \r\n",XXX,AffectedValue,Time_ms);}
    dmx_data[XXX - 1] = AffectedValue;
    updateDMX();
    if(Time_ms>0){wait_us(1000*Time_ms);}
}
if(affichage_texte){printf("Fin de la sequence");}
}
```

6 Bilan des améliorations futures

6.1 Écrire le fichier sur la carte SD depuis la console

La création des fichiers de configuration est une opération fastidieuse. Il serait bien plus simple de développer une fonction permettant de presser un pad et de taper sur l'ordinateur via la console série le canal DMX associé à ce pad. Pour cela, la première étape doit être de récupérer les caractères saisis sur la console, et les enregistrer sur la carte SD. Cette opération est faisable, nous en fournissons un exemple ci-dessous.

Pour ce faire, nous avons développer une petite fonction, appelée par l'interruption que constitue la détection d'un caractère écrit sur la console, capable de récupérer trois chiffres.

```
void ISR_my_pc_reception(void){
    my_pc.read(&data, 1); //data = caractere saisi
    my_pc.write(&data, 1); //on l'affiche car la console ne le fait pas par defaut

    if((data!='\r') && (compteur3 < 3)){ //compteur initialise precedemment pour ne recuperer que trois chiffres
        nombre_complet[compteur3] = data ;
        compteur3 = compteur3 + 1 ;
    }
    else if((data!='\r') && (compteur3 == 3)){
        sprintf(data_s,"Trois chiffres max, on tronque");
        my_pc.write(data_s, strlen(data_s));
    }
    else if(data!='\r'){
        sprintf(data_s,"\r\n");
        my_pc.write(data_s, strlen(data_s));
    }
}
```

Cette fonction est appelée lorsqu'un caractère est détecté sur la console série, et permet de récupérer trois de ces caractères dans la variable `nombre_complet`. Elle est utilisée par la fonction `main` qui suit (qui n'est un main que parce que notre programme n'est qu'une preuve de concept).

```
char data;
char data_s[128];
char nombre_complet[3];
int compteur3;
void ISR_my_pc_reception(void);
int main(){

    data = ' ';
```

```
9       // Try to mount the filesystem
10      int err = fileSystem.mount(&blockDevice);
11      if (!err) {
12          // Open the file /fs/TestEcriture.txt
13          FILE*   f2 = fopen("/fs/TestEcriture.txt", "w+"); //w+ cree le fichier ou l'
        ecrase pour en creer un nouveau -> fichier vierge
14          printf("%s\n", (!f2 ? "Impossible de creer/effacer le fichier:(" : "OKW+"));
15          if (f2) {
16              compteur3 = 0;
17              sprintf(data_s,"Saisir un caractere a enregistrer : ");
18              my_pc.write(data_s, strlen(data_s));
19              my_pc.attach(&ISR_my_pc_reception, UnbufferedSerial::RxIrq);
20              while(compteur3 != 3){ //tant qu'on a pas trois chiffres
21                  wait_us(100);
22              }
23
24              fputs(nombre_complet, f2); // ecriture des trois caracteres dans
        nombre_complet
25              // Exemple [1,2,8] est ecrit dans le fichier "128"
26              //nb_pads = atoi(nombre_complet); //permet de convertir le [1,2,8] en int
         : 128
27              //serait utile si le nombre servirait apres
28              data = ' ';
29              fputs("\r",f2); //on ecrit un saut a la ligne dans le fichier
30              sprintf(data_s,"Saisir un deuxieme caractere a enregistrer : ");
31              compteur3 = 0;
32              my_pc.write(data_s, strlen(data_s));
33              while(compteur3 != 3){
34                  wait_us(100);
35              }
36              fputs(nombre_complet, f2);
37          }
38          // Close the file
39          err = fclose(f2);
40          err = fileSystem.unmount();
41      }
42  }
```

Ce programme fonctionne, et les caractères saisis sont correctement restitués sur le fichier lisible sur la carte SD. En revanche, il ne fonctionne qu'en OS 6 et supérieur, or les précédents programmes étaient écrits en OS 5.x. Il s'avère que la retrocompatibilité est très imparfaite, nous avons cependant réalisé une version mise à jour fonctionnant en OS 6.x du mode 1, qui présente malheureusement plusieurs difficultés. La première d'entre elles est que le fonctionnement devient plus aléatoire (nous avons eu de temps à autre quelques erreurs non reproductibles, sur la lecture du fichier par exemple). L'autre difficulté notable, que nous ne sommes pas parvenu à corriger, est que les signaux DMX sont fréquemment interrompus. Or les projecteurs se mettent en sécurité s'ils ne reçoivent plus de commande, selon un certain délai, délai suffisamment faible dans le cas de la lyre pour qu'elle se mette en sécurité en permanence.

Comme le prouve notre programme de test, il est cependant théoriquement possible de mettre ce système de configuration en place. Il serait ainsi envisageable que la pression sur un bouton de la nucléo lance la fonction de configuration, qui s'en trouverait largement facilitée.

6.2 Enregistrer une séquence en la jouant

Lors de l'utilisation du mode 3 la contrainte actuelle est que l'utilisateur ne peut plus piloter les autres projecteurs tant que la séquence est en cours. Une amélioration future pourra être de laisser les commandes à l'utilisateur lorsque la séquence est active. Pour cela, il faudra revoir la structure même du fichier afin de ne pas mettre en attente la boucle main lorsqu'une séquence est jouée.

De plus, l'écriture d'un fichier séquence peut être long et fastidieux, de ce fait une amélioration pourrait être de donner la possibilité à l'utilisateur de jouer une séquence en direct et de la sauvegarder pour la réutiliser ultérieurement. Ceci permettrait de ne pas avoir à écrire les fichiers contenant les séquences à la main. Toutefois, l'avantage de l'écriture à la main d'un tel fichier permet d'être plus précis

sur les délais et le rythme. Il serait donc agréable d'avoir les deux options.

7 Organisation du travail

7.1 Utilisation de Notion

Pour l'organisation nous avons décidé d'utiliser l'outil Notion dont le lien de notre projet est Notion.

Sur le Notion, nous retrouvons les parties suivantes :

— Tâches : permettant de gérer l'organisation du travail, les délais et la structure du projet et les prises de notes utiles. Contient aussi le diagramme de Gantt : Diagramme de Gantt
— Liens utiles : permettant de faire l'inventaire des documents et lien utiles à tous les membres de l'équipe.
— Documents et Rapport : permettant de synthétiser le fonctionnement et la structure de chaque mode, fonction ou fichier de configuration. C'est une synthèse de ce document.
— Schéma fonctionnel.
— Validation prototype & projet.

7.2 [tripadvisor] Bilan de l'équipe

Je me suis vraiment amusé puisque c'était un travail entre amis. On a tous participé et ce fût vraiment agréable de bosser dans ces conditions. On a vraiment appris plusieurs notions sur comment répondre à des problématiques en groupe. Cependant j'aurais apprécié faire davantage de "réunions" pour assurer un meilleur suivi.

Moi aussi c'était trop cool. J'ai appris à répartir le travail et communiquer de façon claire pour assurer un bon suivi d'équipe. Je recommande. [Avis : 4 étoiles sur 5, achat confirmé]

Fenêtres plein nord, peu recommandé. Ambiance chaleureuse. L'éclairage pilotable est un vrai plus.

Je suis pas d'accord avec la remarque précédente. La lyre faisait mal aux yeux mais j'adore taper des mains pour changer la lumière.

8 Conclusion

Sur la partie technique, notre projet est presque mené à bien. Nous avons respecté le cahier des charges en ayant réalisé les trois modes demandés tout en fournissant à l'utilisateur une facilité d'utilisation. Nous aurions simplement souhaité avec plus de temps et moins de problèmes avec mbed afin de pouvoir regrouper les 3 modes en un seul programme et encore optimiser certaines parties pour une meilleure expérience utilisateur. Pour le plan fonctionnel, comme nous avons pu le montrer lors de la présentation orale des prototypes, nous pouvons correctement modifier l'intensité, la couleur et l'orientation des différents projecteurs.

De plus, les difficultés rencontrées, bien que frustrantes pour l'avancement du projet, nous ont permis de mettre en oeuvre nos capacités de résolution en particulier sur le changement de version de mbed et sur la paramétrage du clavier MIDI.

Enfin sur la partie relationnelle, comme vous pouvez le voir sur les avis, la note moyenne de 4,1/5 témoigne de l'ambiance calme et agréable de la salle S1.1, de sa luminosité, ainsi que des membres y habitant. La collocation avec les membres de cette équipe était forte plaisante et nous regrettons que nos chemins se séparent. En même temps quelle idée d'aller à Saint-Étienne !