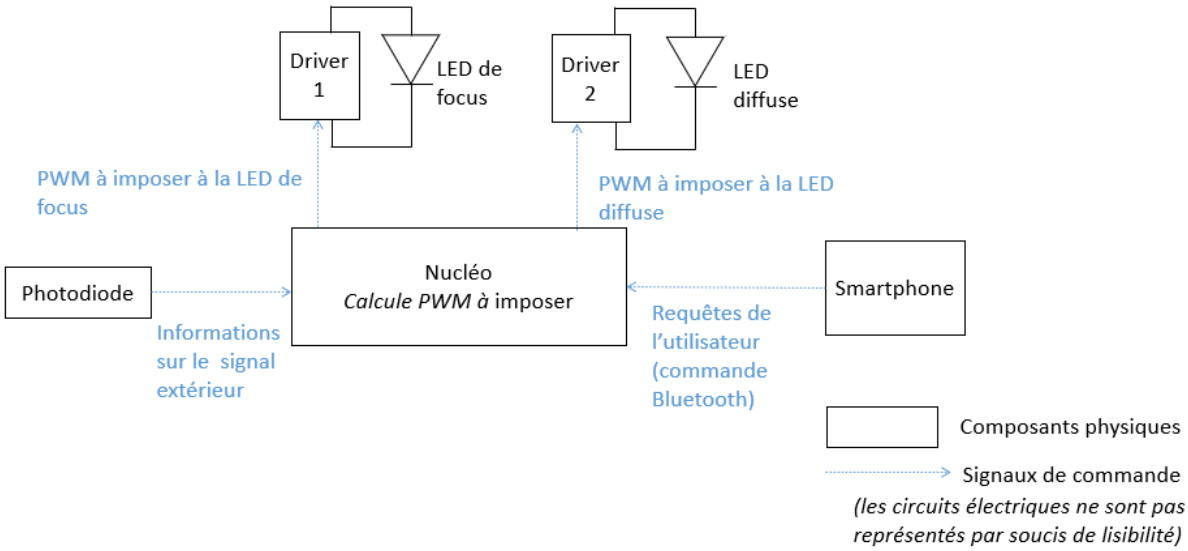
**Louise Devanz, Pierre Senee, Fanny Kalifa, Stéphane Cohen**

# SYSTEME A LED INTELLIGENT

**Introduction :** Notre système est une lampe frontale à LED intelligente. En effet, nous avons modélisé une lampe frontale pouvant illuminer de manière directive ou diffuse selon l'envie de son utilisateur. Ces deux fonctionnalités sont commandées via une interface Bluetooth du smartphone de l'utilisateur.

## I- Architecture générale du système




Tout d'abord, il a fallu penser l'architecture générale de notre système.
L'élément central qui coordonne tout est la Nucléo. Le smartphone envoie la requête de l'utilisateur via une commande Bluetooth à la Nucléo. Celle-ci reçoit aussi des informations sur le signal extérieur. La Nucléo calcule les signal PWM à imposer et le transmet à la LED que l'on veut utiliser (LED de focus ou LED diffuse).

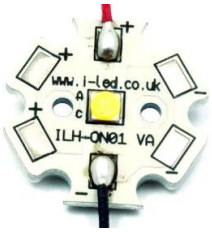
## II- Choix des composants

### 1) Choix des LEDs

La lampe frontale a besoin de 2 LEDs de puissance différente pour fournir un éclairage adapté à courte et longue distance.

a) Choix de la LED pour l'éclairage directif

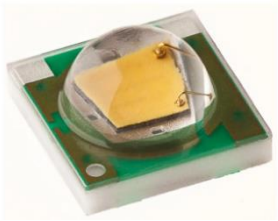
Cette LED permet de fournir un éclairage puissant (280 lm) tout en restant compacte et sans nécessiter de dissipateur de chaleur. (Ce qui est nécessaire pour une utilisation en tant que lampe frontale). Son angle d'émission est de 120 degrés. Pour la rendre plus directive on l'associe à un réflecteur qui permet d'obtenir un faisceau de 35°.




https://fr.rs-online.com/web/p/reseaux-de-led-circulaires/9140337/

b) Choix de la LED pour l'éclairage diffus

Cette LED est moins lumineuse que la précédente (100 lm) elle permet associé à un filtre diffusant d'obtenir un éclairage confortable pour une utilisation à courte distance (par exemple pour la lecture d'une carte).



*https://fr.rs-online.com/web/p/led/8107077*

### 2) Nécessité d'utiliser un driver de LED

Les LED de puissance nécessitent des drivers de LED qui permettent de fournir une valeur de courant précise à la LED. La LED pour l'éclairage directif nécessite 700 mA de courant, celle pour l'éclairage diffus 350 mA. On choisit des drivers correspondant à ces courants.

Éclairage directif : https://fr.rs-online.com/web/p/ci-drivers-de-led/7385711

Éclairage diffus : https://www.digikey.fr/product-detail/fr/recom-power/RCDE-48-0-35/945-RCDE-48-0-35-ND/11307109

De plus les drivers intègrent un contrôle de l'intensité lumineuse de la LED par variation du PWM envoyé à la LED. Ainsi comme on peut le voir dans cet extrait de la datasheet du driver de la LED pour l'éclairage directif, en branchant la broche « PWM DIM » à la carte nucléo, on peut régler le PWM de la LED via la carte nucléo. Le rapport cycle envoyé par la carte nucléo sur la broche « pwm dim » sera le même que celui envoyé par le driver aux LEDs.

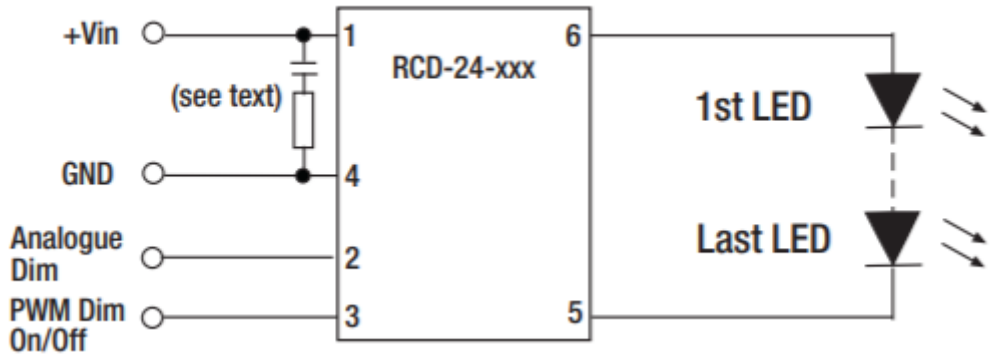



*Fig.1 – Schéma Driver de LED*

3) **Estimation de l'autonomie**

La LED pour l'éclairage directif utilise 700 mA de courant, celle pour l'éclairage diffus 350 mA. En utilisant une batterie rechargeable de charge typique 1.5 Ah on obtient une estimation de 1h30 d'autonomie à pleine puissance. Cette autonomie est satisfaisante sachant que l'utilisateur n'a normalement pas besoin de l'éclairage pleine puissance en permanence.

## III- Circuit électrique

La première partie de ce circuit est composé de la carte nucléo, la LED D1 et de la carte nucléo. En effet, l'utilisateur envoie sa commande (selon l'éclairage désiré) via l'interface Bluetooth de son smartphone sous la forme d'un signal perçu par la LED. La LED D1 joue le rôle de capteur ici. Une fois ce signal reçu, la LED le transmet à la carte nucléo.

Selon celui-ci, la carte nucléo envoie à son tour un signal de commande au Driver 1 (pour un éclairage directif) ou au Driver 2 (pour un éclairage diffus), comme cela est expliqué dans la partie « Asservissement des LEDs ».
Ici, il a fallu utiliser des drivers car la sortie numérique d'une nucléo ne permet pas d'alimenter une LED. En effet, la tension est égale à 0V ou 3,3V (tension de sortie de la nucléo). C'est pourquoi on ajoute un rapport cyclique en sortie de la nucléo (fonctionnalité de la nucléo) ce qui permettra d'éteindre et d'allumer la LED à intervalle de temps régulier (signaux de commande PWM). Ce signal PWM est reçu par l'un des ports du Driver puis est transmis à la LED, amplifié. Le rapport cyclique reste le même mais la tension est plus élevée, ceci permet donc de faire fonctionner les LEDs correctement.

Dans la seconde partie du circuit, on trouver les Drivers 1 et 2. Le Driver correspond à une boîte contenant plusieurs ports.
Il est alimenté avec une tension +/- en entrée. Il reçoit un signal PMW via un de ses ports ($R_{set}$ sur le schéma), comme expliqué précédemment, auquel il faut brancher une résistance en entrée. Ensuite, il renvoie une tension par son port « SW » sur le schéma permettant d'alimenter la LED à laquelle il est branché par les ports « SW » et « LED ». Il ne faut pas oublier, bien-sûr, de le brancher à la masse comme tout composant électrique.

Une batterie est aussi inclue au circuit afin d'alimenter les drivers.

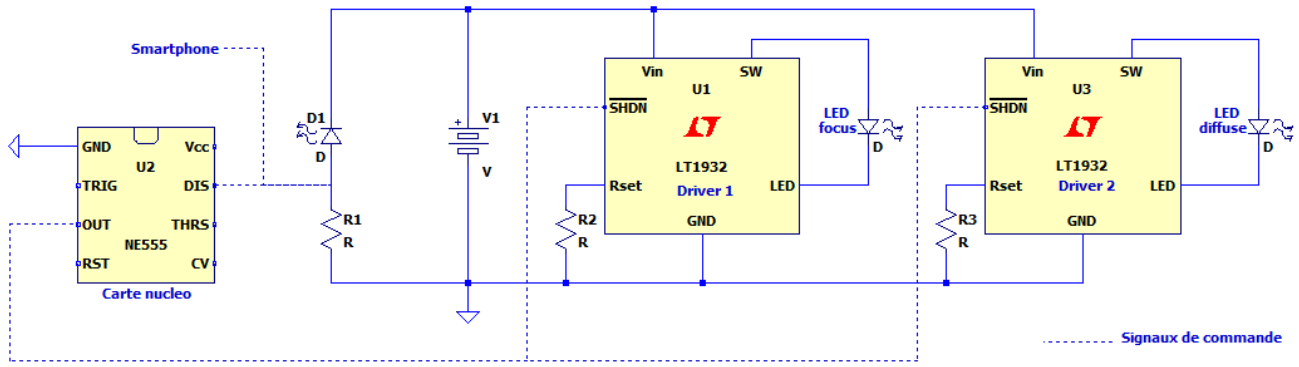



*Fig.2 – Schéma du circuit électrique modélisé avec LTspice*

## IV- Asservissement des LEDs

Les LEDs sont doublement asservies : il y a tout d'abord un choix de la LED à utiliser, puis un asservissement sur l'intensité de la LED.

### a) Réception de la commande Bluetooth

On commence par réceptionner le signal Bluetooth qui impose le mode de fonctionnement (impose la LED à utiliser, l'intensité voulue ou laisse le mode autonome).

### b) Choix de la LED à utiliser

Si la commande Bluetooth impose une LED à utiliser, on sait quelle LED alimenter. Sinon on fait un asservissement autonome du choix de la LED. On relève la valeur de flux reçu/flux envoyé.
Le critère est le suivant : quand on regarde loin, il n'y a pas beaucoup de lumière qui revient, le rapport flux reçu / flux envoyé est petit, tandis que quand on regarde près, il y a beaucoup de lumière qui revient, le rapport flux reçu/flux envoyé est grand.
On choisit donc une valeur de seuil par rapport à laquelle on compare le rapport des flux pour déterminer la LED à asservir en intensité.

### c) Asservissement de l'intensité

Si la commande Bluetooth impose une intensité, on impose directement un PWM relié à la valeur reçue par Bluetooth.
Sinon, on compare la valeur de l'intensité reçue par la photodiode à une valeur seuil et on augmente le PWM ou on le diminue selon la situation.
Voir le code exhaustif en annexe

Remarque :
Nous avons réfléchi à un deuxième critère qui permet de choisir la LED, qui est de regarder la variance du signal reçu. En effet, on peut supposer que si on regarde loin et que l'utilisateur bouge la tête, il balayera différents objets. Ainsi la variance du signal reçu par la photodiode sera élevée.
A contrario, si l'utilisateur lit sa carte par exemple, il regardera toujours le même objet, donc la variance du signal reçu sera alors faible. Nous avons écrit le code permettant l'asservissement des LEDs selon que la variance du signal reçu est supérieure ou inférieure à une valeur seuil. Toutefois, seule l'expérience pourrait nous confirmer l'efficacité de ce critère de sélection, sachant qu'il serait sans doute à affiner car ne s'adapte peut-être pas rigoureusement à toutes les situations. Il peut cependant être une piste pour poursuivre le projet. Nous nous sommes donc concentrés sur le premier critère de sélection, détaillé plus haut.

(Le code associé au mode de fonctionnement autonome, sans commande Bluetooth, sera néanmoins fourni en annexe 1 et 2)

## V- Communication Bluetooth et interface graphique

### 1) Choix du module de connexion bluetooth :

Nous avons choisi le HC 006, il peut être branché directement sur la carte nucléo, sa tension d'alimentation correspond aux broches de sortie de la carte nucléo et peut être trouvé directement par les ports bluetooth d'un terminal android.

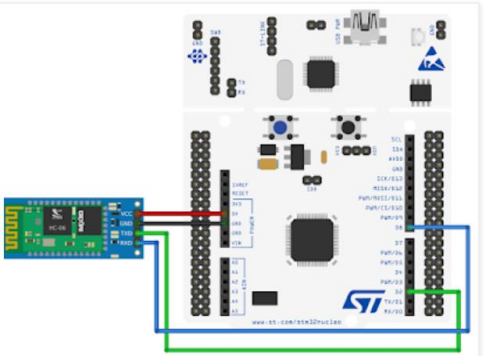


*Fig.3 – Schéma du composant HC 006*

### 2) Interface graphique sous Android

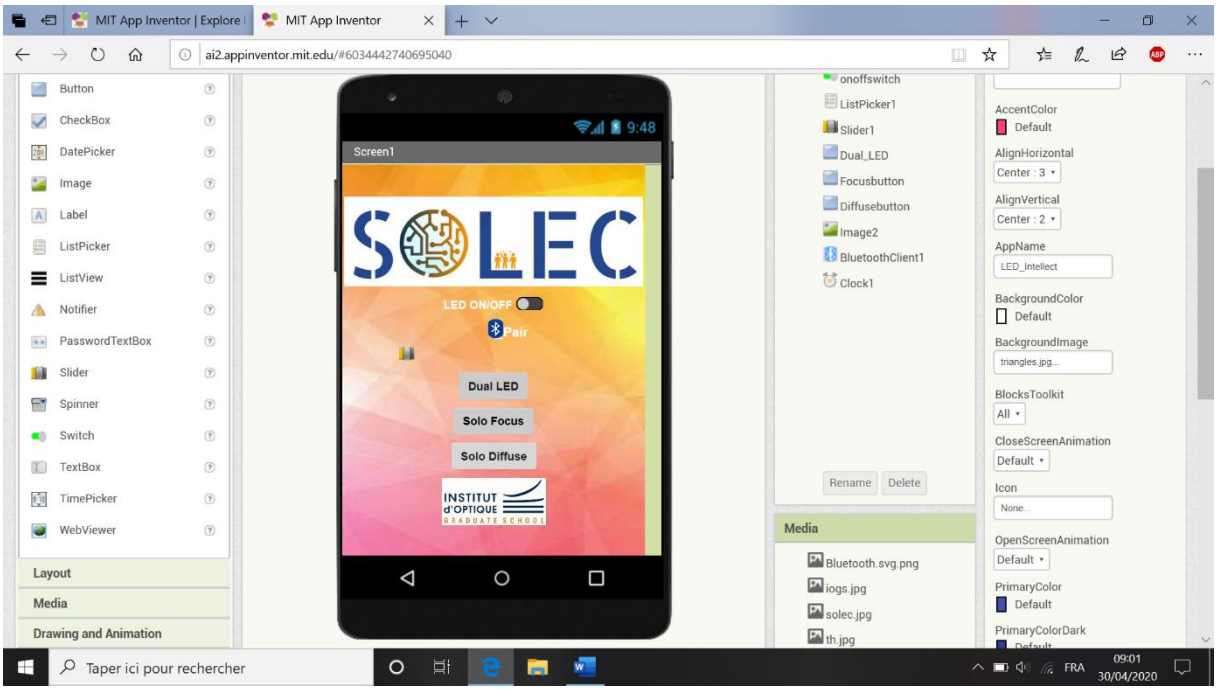



*Fig.4 – Vue sous émulateur de l'application*

Initialisation Bluetooth

L'idée est de rechercher avec le « bluetoothclient » la liste des éléments connectés et de laisser l'utilisateur sélectionner.

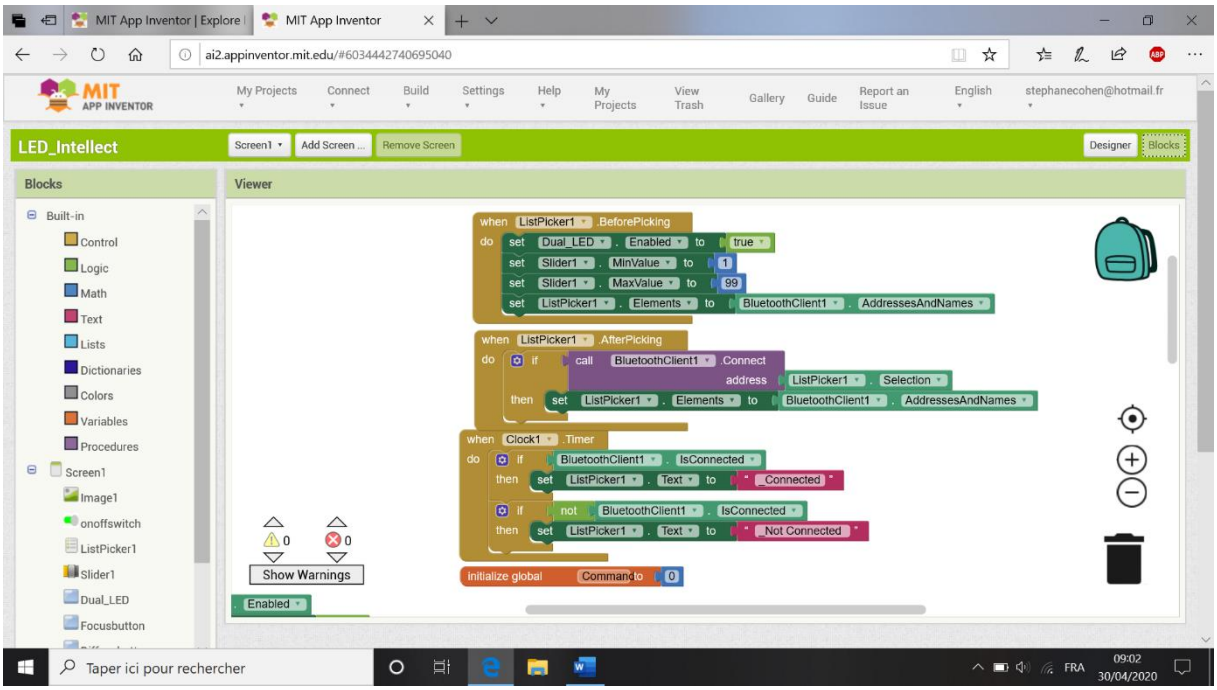



*Fig.5 – Capture d'écran du choix de sélection de l'utilisateur*

<u>Fonctions d'interruption correspondant à la pression de chacun des boutons</u>

Une fonction d'interruption comme celle-ci est créée pour chaque bouton séparément, elles ne sont pas affichées ici car identiques à part l'en tête.

La variable command contient le premier chiffre qui indique le mode de fonctionnement auquel on ajoute un chiffre de 1 à 99 qui permet de régler l'intensité manuellement.

On envoie alors la commande au client bluetooth.

Sur le code mbed du contrôleur ce code est alors traité et transformé en commande PWM pour le driver de la LED.


*Fig.6 – Code transformé en commande PWM*

### 3) Interface graphique sous python

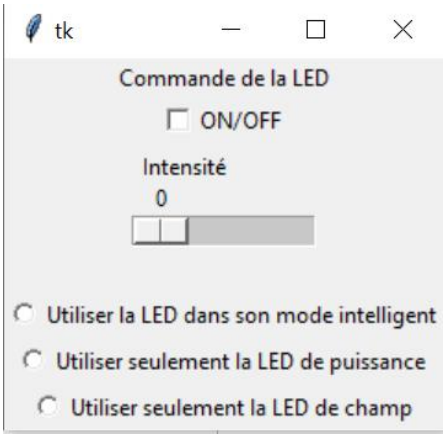



*Fig.7 – Interface vue par l'utilisateur*

Préalablement, il faut connecter son équipement bluetooth et il faut ajouter l'adresse mac du connecteur bluetooth dans le code (voir Annexe 3).

Interface et commande :

A chaque pression de bouton, la fonction sel est appelée, qui lit les valeurs indiquées, les envoie au module bluetooth et affiche à l'écran l'option sélectionnée.

La variable command contient la même information que dans le code sous arduino. Elle contient le premier chiffre qui indique le mode de fonctionnement auquel on ajoute un chiffre de 1 à 99 qui permet de régler l'intensité manuellement.

La commande. pack() permet d'appliquer les modifications et d'ajouter un widget à l'interface (voir code en Annexe 4).

## VI- Retour d'expérience

Dans le contexte du confinement, ce projet a nécessité une organisation particulière.

En effet, les cours n'étant pas en présentiel, il a fallu utiliser des outils de communication dont nous n'aurions pas eu besoin en temps normal. Cependant, une chance que nous avons eu était que Louise, Stéphane et Pierre étaient confinés ensemble tandis que Fanny était de son côté. Le fait d'avoir trois des quatre membres ensemble a considérablement facilité la transmission des informations (un seul appel suffisait pour communiquer tous les quatre). Le transfert d'informations s'est donc fait sans encombre.

Pour ce qui est de l'organisation du travail, il a fallu s'adapter à un nouvel objectif : réaliser un projet virtuel et fictif. Il a été dur au début de se motiver. En effet, le projet n'allant jamais se concrétiser, il était difficile d'en voir l'intérêt. Cependant, tout le monde a continué a joué le jeu et, étant en groupe, nous avons finalement pu nous motiver et fournir quelque chose.

Pour conclure sur ce retour d'expérience, il faut avouer que nous avons tout de même ressenti une certaine frustration à organiser un projet et à réfléchir à chaque côté technique sans pour autant voir notre projet se réaliser sous nos yeux. Il a aussi été frustrant de ne jamais réellement savoir si nos codes fonctionnaient correctement (nous pouvions vérifier leur justesse mais pas s'ils fonctionnaient avec la carte nucléo et réalisaient une action). Cependant, nous gardons un sentiment positif dans l'ensemble, compte tenu des circonstances actuelles, le maintien des cours s'est fait d'une bonne manière et a permis de nous apprendre des choses.

## VII- Bilan des acquis

**Pierre**

Étant en charge du choix des composants constituant la lampe, j'ai dû choisir chaque composant du circuit en fonction de leurs performances. Le choix des LEDs était particulièrement difficile. Le site marchand utilisé (RS component) comporte des milliers de modèles de LEDs et chercher une LED de puissance à la fois compacte et qui ne chauffe pas trop n'était pas facile.

Ce projet m'a aussi permis de comprendre comment sont alimentées les LEDs de puissance et l'importance des drivers à LED pour leur bon fonctionnement.

**Louise**

Lors du projet, j'ai été en charge de la partie codage. J'ai d'une part du me perfectionner au point de vue de la programmation, mais j'ai aussi dû apprendre comment relier les composants physiques au circuit d'électronique de commande.

Pour ce faire, j'ai suivi les tutoriels disponibles sur le site du Lense (notamment "régler l'intensité d'une LED","récupérer un signal analogique" et "faire une action à intervalles réguliers"). Je me suis aussi appuyée sur des programmes que j'avais réalisés lors de TP d'électronique embarquée lors de ma première année à l'Institut d'Optique.

De plus, j'ai appris à intégrer une commande Bluetooth à mon programme, en utilisant le site https://electroniqueamateur.blogspot.com/2017/07/module-bluetooth-hc-06-et-stm32-nucleo.html .

Enfin, j'ai du apprendre par moi-même le fonctionnement des driver en m'appuyant sur les datasheet (https://fr.rs-online.com/web/p/ci-drivers-de-led/7385711/ et https://www.digikey.fr/product-detail/fr/recom-power/RCDE-48-0-35/945-RCDE-48-0-35-ND/11307109) des composants retenus, notamment comprendre comment ils sont

commandés et comment ils restituent la commande aux LED, afin que je puisse envoyer des instructions adéquates aux drivers à partir de la carte Nucléo et créer l'architecture du circuit électrique.

**Fanny**

Lors de ce projet, j'étais en charge de la conception du circuit électrique.
Pour ce faire, j'ai principalement étudié le logiciel « LTspice ». J'ai suivi tous les tutoriels proposés (ils allaient plus loin que ce dont j'avais besoin mais ce ne sont pas des acquis perdus).
De plus, la conception du circuit électrique m'a permis de comprendre le concept de driver et leur utilisation avec des LEDs.

**Stéphane**

Ce projet m'a permis de rafraichir mes souvenirs de Python, ainsi que de découvrir la bibliothèque tkinter permettant des interfaces graphiques.
De plus j'ai pu me familiariser avec les protocoles de connexion Bluetooth. J'ai à la fois étudié la connexion depuis l'interface Python et depuis un appareil Android.
J'ai aussi découvert le MIT AppInventor qui permet de concevoir des applications sous Android. Cet outil est un outil très simple d'utilisation et n'offre donc pas un très grand panel de possibilités mais suffit pour une application mobile pilotant une LED par exemples.
La conception des interfaces a nécessité l'étroite collaboration avec Louise Devanz qui a écrit les codes du microcontrôleur afin d'en assurer la compatibilité et a permis des apprentissages des capacités d'organisations communes.

## Conclusion :

Au niveau des compétences non strictement techniques, ce projet nous a appris à diviser le travail et à nous appuyer sur les compétences nouvellement acquises par les autres membres de l'équipe afin d'aboutir au projet final (notamment car nous avons séparé la partie codage, réalisation de l'interface Bluetooth, conception du circuit électrique et choix des composants). Pour ce faire, la maîtrise de l'outil de collaboration BaseCamp nous a été utile.

Finalement, ce projet s'est relativement passé bien dans l'ensemble selon nous. Evidemment, il aurait été plus intéressant de concevoir notre prototype, mais compte tenu des circonstances, nous sommes satisfaits de ce que nous avons pu réaliser.

## VII- Annexe

Annexe 1 : Code d'asservissement des LEDs pour le 1er critère de sélection : comparaison du rapport flux reçu/flux envoyé

```
#include "mbed.h" //compare à intervalles réguliers la valeur renvoyée par la photodiode à un seuil,
//et ajuste l'intensité de la LED en conséquence et choisit la LED à allumer

Ticker toggle_signal_ticker; // Déclaration du ticker
AnalogIn signal_photodiode(A0);  // Déclaration de l'entrée analogique (signal photodiode)
PwmOut LED_focus_pwm(D10);  //sortie branchée au driver associé à la LED de focus (pour regarder au loin)
PwmOut LED_diffus_pwm(D9); //sortie branchée au driver associé à la LED diffuse (pour regarder près)
Serial bluetooth(D8, D2); //entrée de la commande bluetooth


double mesure_photodiode;  // variable de lecture du signal
void asservissement_diode (void); //déclaration de la variable associée au ticker

double marge_rapport; //on définit une marge d'erreur autour du seuil (seuil_rapport) pour que la lumière ne vacille pas si on est proche du seuil
double marge; //on définit une marge d'erreur autour du seuil pour que la lumière ne vacille pas si on est proche du seuil
double rapport_seuil; //valeur du seuil du rapport du flux reçu sur flux sortant à l'itération précédente
double seuil_pres; //seuil en intensité électrique du signal de la photodiode acceptable pour voir de près
double seuil_loin; //seuil en intensité électrique du signal de la photodiode acceptable pour voir loin
double periode_LED_focus;
double periode_LED_diffus;
double periode_echantillonnage;
double PWM;
double PWM_utilisateur; //PWM imposée par l'utilisateur en mode "commande d'intensité"
double T[2]={0,0};//initalisation du tableau servant à comparer le flux reçu au flux émis à l'exécution précédente de la boucle
int c= bluetooth.getc(); // commande bluetooth qui comprend en c[0] le mode de fonctionnement et en c[1] la valeur du seuil, en PWM.
// Par défaut, si une valeur est à zéro, ça veut dire qu'on laisse le pilotage automatique, alors on compare à des valeurs seuil
//de référence sans demander à l'utilisateur

int main() {

toggle_signal_ticker.attach(&asservissement_diode, periode_echantillonnage); //choisit la période d'échantillonnage
```

```
    while(1) {
     }}


void asservissement_diode() {
    LED_diffus_pwm.period_ms(periode_LED_diffus); //définit la période de la LED diffuse
    LED_focus_pwm.period_ms(periode_LED_focus);
    mesure_photodiode = signal_photodiode.read_u16();  // sur 12 bits MSB
    T[0]=T[1]; //on décale progressivement les valeurs du tableau
    T[1]=PWM;
    c=600; //si le blutooth ne revoit pas de commande, on choisit le mode totalement autonome
    c=bluetooth.getc();

if (c==600) //cas du pilotage totalement autonome, intensité et choix de la LED asservis
{

    if (((mesure_photodiode/T[0])>(rapport_seuil - marge_rapport)) && PWM<1) //c'est le cas où on est près, on utilise LED diffuse
    { LED_focus_pwm.write(0); //on éteint l'autre LED
            if (mesure_photodiode<(seuil_pres - marge) && PWM<1) { //cas où la luminosité est insuffisante
            LED_diffus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(seuil_pres + marge) && PWM<1) { //cas où la luminosité est trop forte
            LED_diffus_pwm.write(PWM-0.1);
            }
    }

    if (((mesure_photodiode/T[0])>(rapport_seuil - marge_rapport)) && PWM<1) //c'est le cas où on est loin, on utilise LED focus
            { LED_diffus_pwm.write(0);
            if (mesure_photodiode<(seuil_loin - marge) && PWM<1) { //cas où la luminosité est insuffisante
```

```
            LED_focus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(seuil_loin + marge) && PWM<1) { //cas où la luminosité est trop forte
            LED_focus_pwm.write(PWM-0.1);
            } }

        }

if (c==500) // on choisit la LED diffuse et l'intensité est asservie par rapport à la valeur constructeur
{
LED_focus_pwm.write(0);

            if (mesure_photodiode<(seuil_pres - marge) && PWM<1) { //cas où la luminosité est insuffisante
            LED_diffus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(seuil_pres + marge) && PWM<1) { //cas où la luminosité est trop forte
            LED_diffus_pwm.write(PWM-0.1);
             }
        }


if (c==400) // on choisit la LED diffuse et l'intensité est asservie par rapport à la valeur constructeur
        {
LED_diffus_pwm.write(0);
            if (mesure_photodiode<(seuil_loin - marge) && PWM<1) { //cas où la luminosité est insuffisante
            LED_focus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(seuil_loin + marge) && PWM<1) { //cas où la luminosité est trop forte
            LED_focus_pwm.write(PWM-0.1);
            }
            }

if (200<c<301) // choix de la LED automatique et intensité demandée par l'utilisateur
             {
       PWM_utilisateur=(c-200)/100;


        if (((mesure_photodiode/T[0])>(rapport_seuil - marge_rapport)) && PWM<1) //c'est le cas où on est près, on utilise LED diffuse
         { LED_focus_pwm.write(0);
            if (mesure_photodiode<(PWM_utilisateur - marge) && PWM<1) { //cas où la luminosité est insuffisante par rapport à la valeur
            //demandée par l'utilisateur
            LED_diffus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(PWM_utilisateur + marge) && PWM<1) { //cas où la luminosité est trop forte
            LED_diffus_pwm.write(PWM-0.1);
            } }

        if (((mesure_photodiode/T[0])>(rapport_seuil - marge_rapport)) && PWM<1) //c'est le cas où on est loin, on utilise LED focus
            { LED_diffus_pwm.write(0);
            if (mesure_photodiode<(PWM_utilisateur - marge) && PWM<1) { //cas où la luminosité est insuffisante
            LED_focus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(PWM_utilisateur + marge) && PWM<1) { //cas où la luminosité est trop forte
            LED_focus_pwm.write(PWM-0.1);
            } }

        }
```

```
if (100<c<201) //c'est le cas où on est près, on utilise LED diffuse et l'intensité est imposée par l'utilisateur
{
PWM_utilisateur=(c-100)/100;
LED_focus_pwm.write(0);

    if (mesure_photodiode<(PWM_utilisateur - marge) && PWM<1) { //cas où la luminosité est insuffisante par rapport à la valeur
    //demandée par l'utilisateur
             LED_diffus_pwm.write(PWM+0.1);
             }

    if (mesure_photodiode>(PWM_utilisateur + marge) && PWM<1) { //cas où la luminosité est trop forte
             LED_diffus_pwm.write(PWM-0.1);
             }
             }

if (0<c<101) //c'est le cas où on est loin, on utilise LED de focus et l'intensité est imposée par l'utilisateur
{LED_diffus_pwm.write(0);
PWM_utilisateur=c/100;

    if (mesure_photodiode<(PWM_utilisateur - marge) && PWM<1) { //cas où la luminosité est insuffisante par rapport à la valeur
    //demandée par l'utilisateur
    LED_focus_pwm.write(PWM+0.1);
    }

    if (mesure_photodiode>(PWM_utilisateur + marge) && PWM<1) { //cas où la luminosité est trop forte
    LED_focus_pwm.write(PWM-0.1);
    }
}

if (c==0)// on éteint les deux lampes

{LED_focus_pwm.write(0);
LED_diffus_pwm.write(0);
}
}
```

## Annexe 2 : Code d'asservissement des LEDs pour le 2ème critère de sélection : calcul de la variance du signal

```
#include "mbed.h" //compare à intervalles réguliers la valeur renvoyée par la photodiode à un seuil, et ajuste l'intensité de la LED
//en conséquence et choisit la LED à allumer

Ticker toggle_signal_ticker; // Déclaration du ticker
AnalogIn signal_photodiode(A0);  // Déclaration de l'entrée analogique (signal photodiode)
PwmOut LED_focus_pwm(D10);  //sortie de la LED de focus (pour regarder au loin)
PwmOut LED_diffus_pwm(D9); //sortie de la LED diffuse (pour regarder près)

double mesure_photodiode;  // variable de lecture du signal
void asservissement_diode (void); //déclaration de la variable associée au ticker

double marge_variance; //on définit une marge d'erreur autour de la variance seuil pour que la lumière ne vacille pas si on est proche du seuil
double marge; //on définit une marge d'erreur autour du seuil pour que la lumière ne vacille pas si on est proche du seuil
double seuil_variance; //valeur du seuil de la variance
double seuil_pres; //seuil en intensité électrique du signal de la photodiode acceptable pour voir de près
double seuil_loin; //seuil en intensité électrique du signal de la photodiode acceptable pour voir loin
double periode_LED_focus;
double periode_LED_diffus;
double periode_echantillonnage;
double PWM;
double T[2]={0,0};//initalisation du tableau servant à comparer le flux reçu au flux émis à l'exécution précédente de la boucle
double V[10]={0,0,0,0,0,0,0,0,0,0};//initalisation du tableau servant à faire la moyenne et la variance
int i; //Déclaration de la variable de boucle
double M_volt; //variable de moyenne, convertie en volt
double M;  //variable de moyenne
int sum; //variable somme servant pour calculer la moyenne
int sum1; //variable somme servant pour calculer la variance
double variance;
int main() {

toggle_signal_ticker.attach(&asservissement_diode, periode_echantillonnage); //choisit la période d'échantillonnage
```

```
    while(1) {
     }}


double calcul_variance(){
    mesure_photodiode = signal_photodiode.read_u16();  // sur 12 bits MSB
        sum=0;
        V[0]=V[1]; //on décale progressivement les valeurs du tableau
        V[1]=V[2];
        V[2]=V[3];
        V[3]=V[4];
        V[4]=V[5];
        V[5]=V[6];
        V[6]=V[7];
        V[7]=V[8];
        V[8]=V[9];
        V[9]=mesure_photodiode; //on rajoute comme dernière valeur la valeur acquise

        for (i = 0; i < 9; ++i) { //permet de calculer la moyenne

        sum += V[i];
    }

    M = sum / 10; //moyenne du tableau

    for (i = 0; i < 9; i++)
    {
        sum1 = sum1 + (V[i] - M)*(V[i] - M);
    }
    variance = sum1 /10; //variance du tableau

    return variance;
}


void asservissement_diode() {
    variance=calcul_variance();
    LED_diffus_pwm.period_ms(periode_LED_diffus); //définit la période de la LED diffuse
    LED_focus_pwm.period_ms(periode_LED_focus);
    mesure_photodiode = signal_photodiode.read_u16();  // sur 12 bits MSB
    T[0]=T[1]; //on décale progressivement les valeurs du tableau
    T[1]=PWM;

         if ((variance>(seuil_variance - marge_variance)) && PWM<1) //c'est le cas où on est près, on utilise LED diffuse
          {
            if (mesure_photodiode<(seuil_pres - marge) && PWM<1)
            { //cas où la luminosité est insuffisante
            LED_diffus_pwm.write(PWM+0.1);
            }

            if (mesure_photodiode>(seuil_pres + marge) && PWM<1)
            { //cas où la luminosité est trop forte
            LED_diffus_pwm.write(PWM-0.1);
            }
            }

         if ((variance>(seuil_variance - marge_variance)) && PWM<1) //c'est le cas où on est loin, on utilise LED focus
            {
            if (mesure_photodiode<(seuil_loin - marge) && PWM<1)
            { //cas où la luminosité est insuffisante
            LED_focus_pwm.write(PWM+0.1);
            }
```

Annexe 3 : Code interface graphique sous Python dans lequel on connecte l'équipement bluetooth et on ajoute l'adresse mac du connecteur bluetooth

```
#connexion bluetooth
#1e etape lister les appareils présents
import bluetooth
import numpy as np
from tkinter import *
nearby_devices = bluetooth.discover_devices(lookup_names=True)
for addr, name in nearby_devices:
    print("  %s - %s" % (addr, name))

target_name = "%s " %(name(1))   #on sélectionne le premier peripherique,
#les périphériques étant listés par intensité du signal reçu,
#il suffit d approcher la lampe  a l'initialisation
target_address = None
for bdaddr in nearby_devices:
    if target_name == bluetooth.lookup_name( bdaddr ):
        target_address = bdaddr
        break

if target_address is not None:
    print ("found target bluetooth device with address ", target_address)
else:
    print ("could not find target bluetooth device nearby")
#préparation à la communication
    serverMACAddress = 'adresse mac du module HC006 à remplacer par la vraie valeur'
port = 1
s = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
s.connect((serverMACAddress, port))
```

Annexe 4 : Code avec commande .pack() permettant d'appliquer les modifications et d'ajouter un widget à l'interface

```

#interface graphique
command=600;
fenetre = Tk()
label = Label(fenetre, text="Commande de la LED")
label.pack()
power=IntVar;
power = Checkbutton(fenetre, text="ON/OFF")
power.pack()
intensite = DoubleVar()
scale = Scale(fenetre,label="Intensité",orient=HORIZONTAL,variable=intensite)
scale.pack()
value = IntVar()
#la fonction sel s'active à chaque pression de bouton, affiche le mode sélectionné en message,
#envoie le message au module bluetooth et ferme la connexion

def sel():
    selection = "Vous avez selectionné le mode " + str(value.get())
    label1.config(text = selection)
    command="%s " %(100*str(value.get()+scale.get()) #La variable command contient le premier chiffre
    # qui indique le mode de fonctionnement
    # auquel on ajoute un chiffre de 1 à 99 qui permet de régler l'intensité manuellement.
    s.send(command)
    s.close()
label1=Label(fenetre)
label1.pack()

modenormal=Radiobutton(fenetre, text="Utiliser la LED dans son mode intelligent",variable=value, value=1,command=sel)
modenormal.pack()
modepuissance=Radiobutton(fenetre, text="Utiliser seulement la LED de puissance",variable=value, value=2,command=sel)
modepuissance.pack()
modegrandchamp=Radiobutton(fenetre, text="Utiliser seulement la LED de champ",variable=value, value=3,command=sel)
modegrandchamp.pack()
fenetre.mainloop()
```