Rapport Technique: Système à LEDs intelligent

Angèle Denis, Emma Aoustin, Pierre Lebegue, Tom Dupuis
Projet PROTIS 2022

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech
LEnsE Laboratoire d'Enseignement Expérimental

Sommaire

I. Introduction

Le produit est un système à LED intelligent qui permet d'asservir l'intensité en fonction de son utilisation. L'utilisateur peut contrôler la luminosité à l'aide d'une application mais la régulation pourra être automatique. La contrainte que nous avions était une contrainte de temps : le système doit tenir plusieurs heures en autonomie. Nous avons donc dû quantifier la consommation des différents éléments présents dans le système (LEDs et microcontrôleur principalement).

Les performances demandées sont les suivantes :

- Flux lumineux qui doit atteindre les 500 lm
- Utilisation possible de plusieurs LED pour varier les modes d'éclairage (large ou focalisé)
- Plusieurs modes d'éclairage à proposer

Figure n°1 : objectif final, prêt pour commercialisation

II. Caractéristiques des éléments du montage

a. *LEDs*

Nous avons choisi pour notre système d'utiliser deux types de LEDs : une centrale plus puissante et deux autres LEDs secondaires pour les côtés. Nous avons donc, dans un premier temps, mesurer les caractéristiques de nos LEDs (figure n°2).

LED centrale : https://asset.conrad.com/media10/add/160267/c1/-/gl/001503528DS01/fiche-technique-1503528-barthelme-led-high-power-blanc-chaud-6-w-510-lm-130-1800-ma-61003528.pdf

LEDs secondaires : https://docs.rs-online.com/290b/0900766b8141f571.pdf

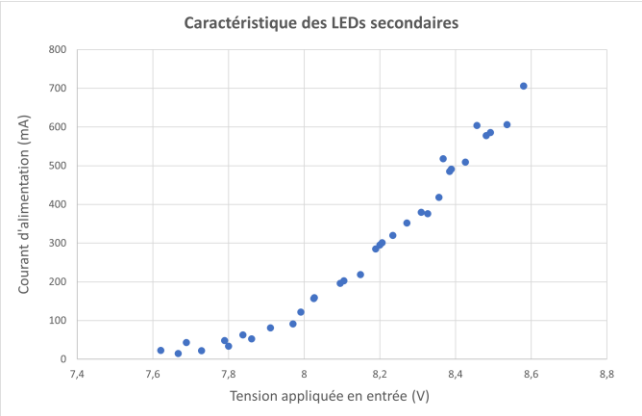

Figure n°2 : caractéristique des LEDs secondaires

b. *Réflecteurs*

Pour avoir une utilisation optimale des LEDs, nous avons choisi de les associer à des réflecteurs. Nous avons également utilisé une lentille asphérique au niveau de la LED pour avoir un flux plus directif pour notre LED centrale.

https://docs.rs-online.com/023c/0900766b814242a9.pdf

c. ***Photodiode***

Notre système étant asservi en luminosité, nous avions besoin de mesurer le flux ambiant de l'environnement, donc avons donc choisi une photodiode dans le domaine visible. La photodiode mesure la luminosité, nous réalisons ensuite une moyenne glissante sur 10 mesures réalisé à l'aide d'une fonction d'interruption appelé à intervalle régulier de 10ms.

https://docs.rs-online.com/67af/0900766b808b25c4.pdf

d. ***Accéléromètre***

Nous avons décidé d'insérer dans le mode automatique, une partie qui détermine le nombre de LEDs allumées lors de l'utilisation du produit. Le but était d'avoir une lumière directive lorsque nous étions en mouvement, sur le modèle de la vue (le champ se retreint proportionnellement à la vitesse), et d'avoir une lumière diffuse à l'arrêt. Nous avons donc, pour cela, utiliser un accéléromètre.

L'accéléromètre que nous avons choisi d'utiliser est un accéléromètre ADXL335.

L'accéléromètre mesure, comme son nom l'indique, l'accélération selon chaque composante de l'espace : x,y et z. Dans un premier temps, et nous avons trouvé ça suffisant pour l'utilisation que nous en avions, nous avons utilisé une seule composante de l'accéléromètre : celle selon z. L'accéléromètre délivre une tension lorsque le composant est mis en mouvement. Pour quantifier cette mesure, nous ne pouvions pas uniquement la prendre brute et l'utiliser dans le code. En effet, il fallait déterminer les variations de tension correspondantes aux mouvements. Nous avons donc calculer l'écart-type sur les 10 valeurs glissantes. Nous avons, suite de nombreux tests, déterminer un seuil pour cet écart-type au-delà duquel nous considérons être dans un mode "mouvement".

https://www.analog.com/media/en/technical-documentation/data-sheets/adxl335.pdf

III. Le montage électrique

Nous avons réalisé trois branchements électriques différents avec trois alimentations séparées : deux alimentations sont branchées sur la carte Nucléo (3,3V et 5V) et une autre alimentation est réalisée grâce une batterie.

Le premier montage (figure n°3) permet de commander les trois LEDs et d'alimenter la photodiode. Les LEDs que nous utilisons sont des LEDs de puissance, et nous voulons pouvoir commander leurs intensités via le microcontrôleur, grâce à des commandes PWM (Pulse Width Modulation). Le principe des PWM est d'appliquer une suite d'états discrets en choisissant les rapports cycliques souhaités, on peut obtenir la valeur d'intensité souhaitée en ne regardant que la valeur moyenne du signal. Afin de pouvoir commander les LEDs de puissance, nous avons dû utiliser des transistors. La carte Nucléo n'est pas capable de fournir beaucoup de courant pour commander les LEDs de puissance, alors les transistors servent à réguler le flux de courant électrique et qui permet d'amplifier un courant. Nous avons vu que la tension seuil de nos LEDs était de 7,6V. On a constaté qu'une alimentation batterie de 7,2V (tension délivrée à mi-décharge) fonctionnait. Cependant, nos 3 LEDs n'ont pas les mêmes caractéristiques, la LED centrale (LED 2) était beaucoup plus brillante lorsqu'elle était alimentée directement avec 7,2 V, nous avons donc rajouter une résistance afin de pouvoir avoir un flux lumineux homogène en sortie et de ne pas être ébloui par la LED centrale.

Pour la photodiode, nous avons décidé de l'alimenter grâce à la batterie. Nous utilisons aussi un filtre passe-bas RC avant d'atteindre la carte nucléo. Le filtrage et le moyennage (réalisé par le code du microcontrôleur)

permettent de s'affranchir du bruit et d'une éventuelle variation d'éclairage dû au 50Hz des éclairages commerciaux ou dû aux commandes PWM.

De plus, nous devons alimenter la carte Bluetooth avec une tension de 5 V, pour cela nous n'avons pas besoin d'une alimentation externe, nous pouvons simplement utiliser les 5 V de la carte Nucléo (figure n°5). La sortie de la carte Bluetooth doit aussi être branchée au microcontrôleur (branches D2 et D8).

Enfin, nous avons aussi le branchement de l'accéléromètre a réalisé (figure n°4). Il est alimenté avec une tension de 3,3 V, il suffit de prendre les 3,3 V de la carte Nucléo. En sortie de l'accéléromètre, on peut avoir l'accélération selon les directions x, y et z. Nous avons choisi de récupérer seulement l'accélération selon z, que l'on récupère sur une entrée du microcontrôleur.

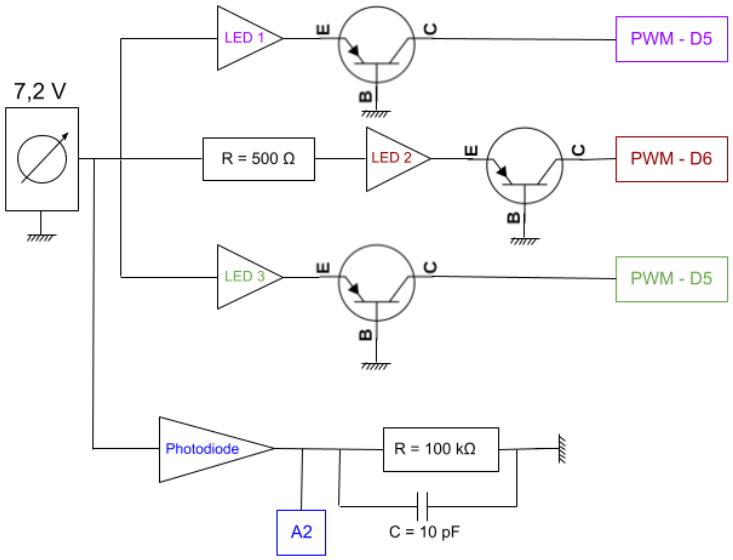

Figure n°2 : schéma électrique du montage avec les 3 LEDs et la photodiode

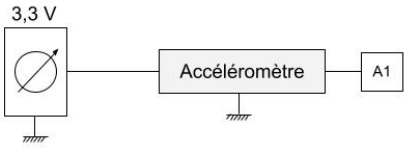

Figure n°3 : schéma électrique de l'accéléromètre

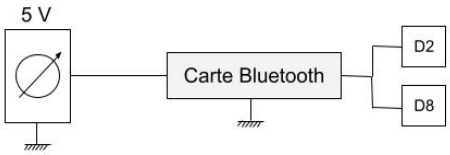

Figure n°4 : schéma électrique de la carte Bluetooth

Ensuite, afin de pouvoir débrancher notre prototype de l'ordinateur et pouvoir assurer une autonomie de notre système afin qu'une portabilité, nous devons alimenter la carte Nucléo avec notre batterie. Pour cela, il faut réaliser une petite manipulation sur la carte Nucléo : on doit mettre un cavalier sur les positions 1 et de 2 de JP5. Il faut faire attention car quand nous sommes dans cette configuration, on ne peut pas brancher la carte à l'ordinateur. Si on veut envoyer un code sur la carte, il faut réaliser la manipulation inverse (cavalier sur les positions 2 et 3 de JP5).

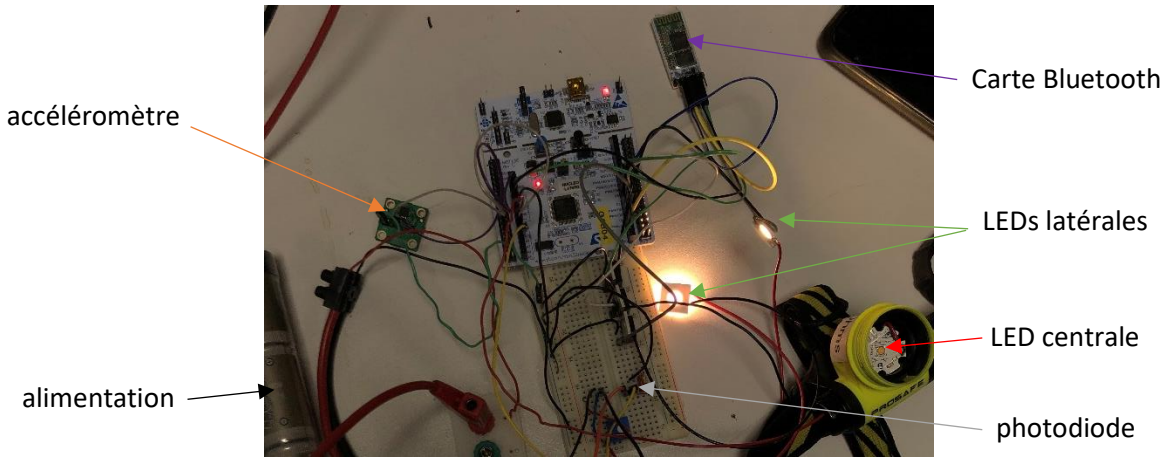

Figure n°6 : photo du montage électrique de notre prototype

IV. Notre application pour Android

L'application a été développée sur MIT App Inventor, un site internet gratuit. L'application a été programmée pour convenir au code MBED donné en annexe.

Elle comporte :

- Un bouton marche/arrêt qui permet d'allumer ou d'éteindre le système
- Un mode manuel qui permet de régler le nombre de LEDs allumées (à l'aide du bouton Mouvement/ Arrêt) et la luminosité des LEDs à l'aide d'un curseur
- Un mode automatique où l'on peut choisir la consigne de luminosité pour les LEDs à l'aide d'un curseur

La programmation sur MIT App Inventor se fait par bloc et par élément (un bloc représente un bouton ou un curseur). Les captures suivantes vous permettent de comprendre comment fonctionne le site, et de reproduire l'application sur un compte MIT App Inventor. Quelques éléments de graphisme ont été ajouté (disparition de bouton lorsqu'on quitte un mode et image).

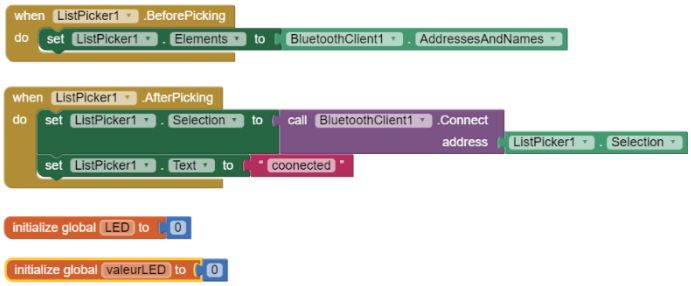

Figure n°7 : Capture d'écran MIT App Inventor : initialisation de l'application

Cette première partie du code de l'application permet d'initialiser la commande Bluetooth (appelée BluetoothClient1) et d'afficher le message « coonected » lorsque la connexion est établie. Elle permet également d'éteindre les LEDs en initialisation.

Sur la page suivante figurent les captures d'écran du site internet MIT App Inventor avec les programmations par blocs de chaque partie.

Le rapport technique détaillera la figure 8c (partie du mode Manuel/ Automatique), les autres blocs étant codés sur le même principe.

On rentre dans un bloc lorsque que l'on appuie sur le bouton : on passe soit du mode *« Global = 1 »* au mode *« Global = 2 »*. La variable *« Global »* permet simplement de mémoriser le mode. Une fois dans une des deux boucles, on modifie le bouton sur l'interface de l'application : si on passe du mode *Manuel* au mode *Automatique* alors le bouton est modifié (nom du mode et couleur). On modifie ensuite la variable *« Global »* pour mémoriser le changement. On envoie ensuite, via la liaison Bluetooth, l'information que la variable correspondant au mode a changée : pour le mode Manuel/Automatique la variable est *« M »* et on la met à 1 lorsque le mode mouvement est sélectionné par l'utilisateur et à 0 si c'est le mode arrêt qui a été choisi. Ensuite, trois lignes sont attribuées au graphisme. En effet, dans le mode automatique, le curseur correspondant à la luminosité des LEDs dans le mode Manuel n'est plus utile donc on le cache (ici cela correspond à *« set HorizontalArrangement2 Visible to False »*). Au contraire, le curseur correspondant à la consigne donnée à l'asservissement doit être visible (*« set HorizontalArrangement3 Visible to True »*). Le bouton Arrêt/Mouvement doit aussi être caché. (*« set Bouton_Mouvement_Arrêt Visible to False »*).

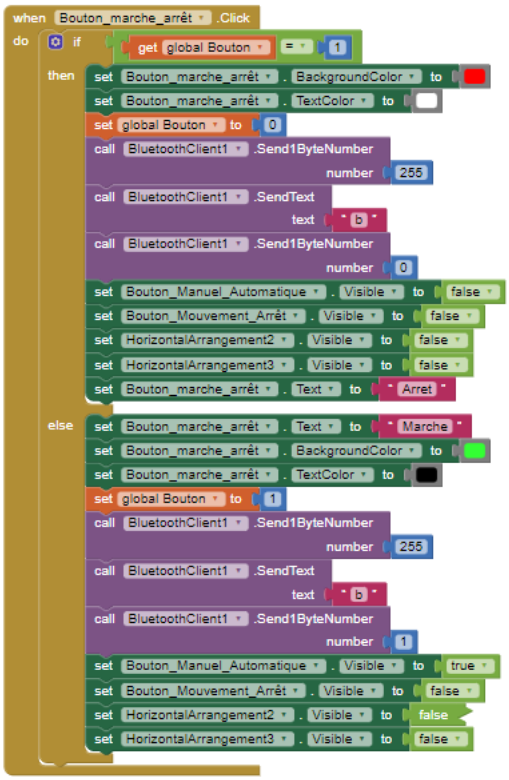

Figure 8a : Programmation de l'application bouton marche/arrêt

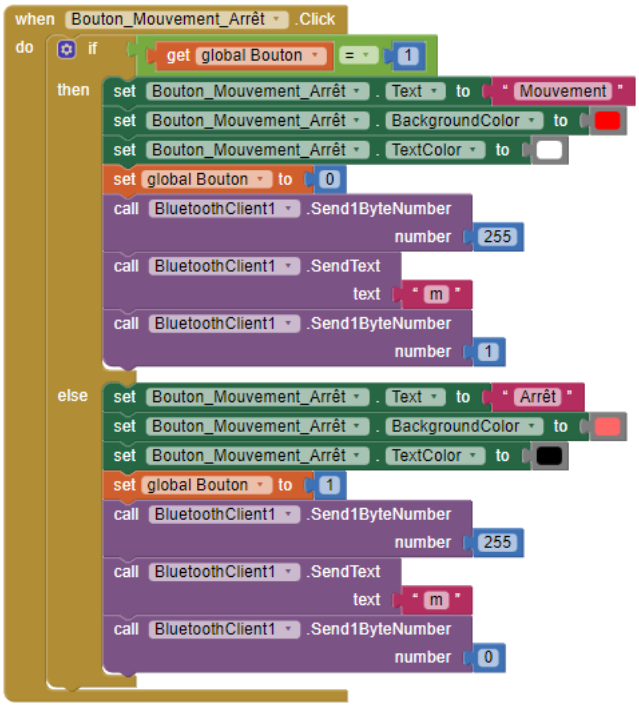

Figure 8b : Programmation de l'application bouton mouvement/arrêt

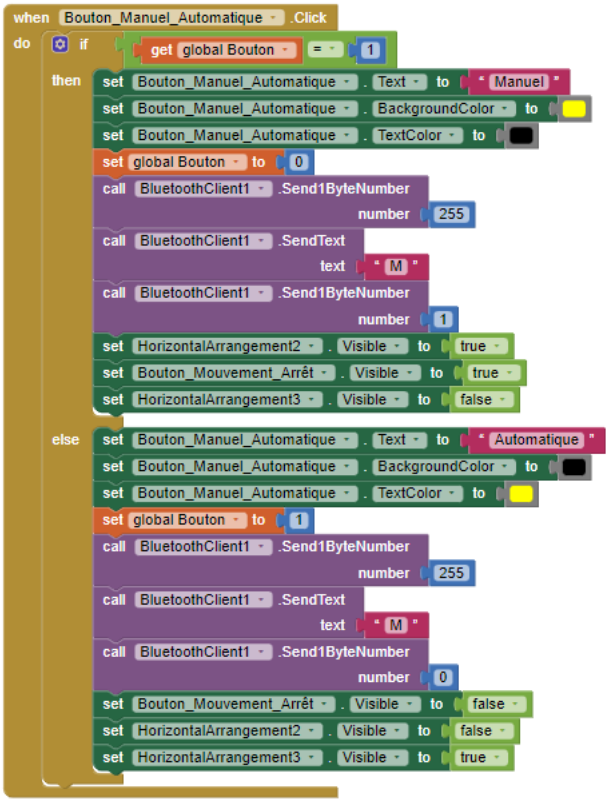

Figure 8c : Programmation de l'application bouton manuel/automatique

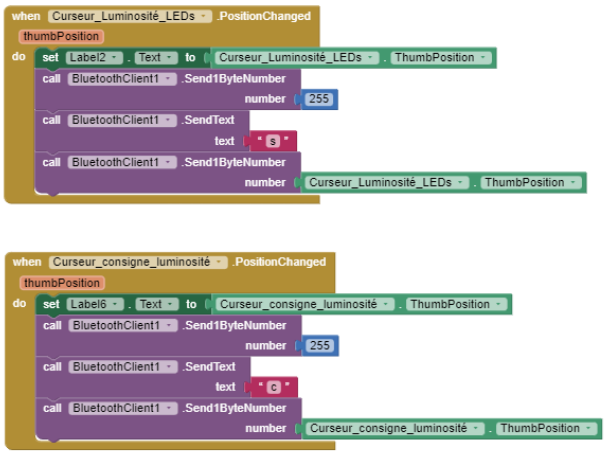

Figure 8d : Programmation de l'application des curseurs

V. Fonctionnement

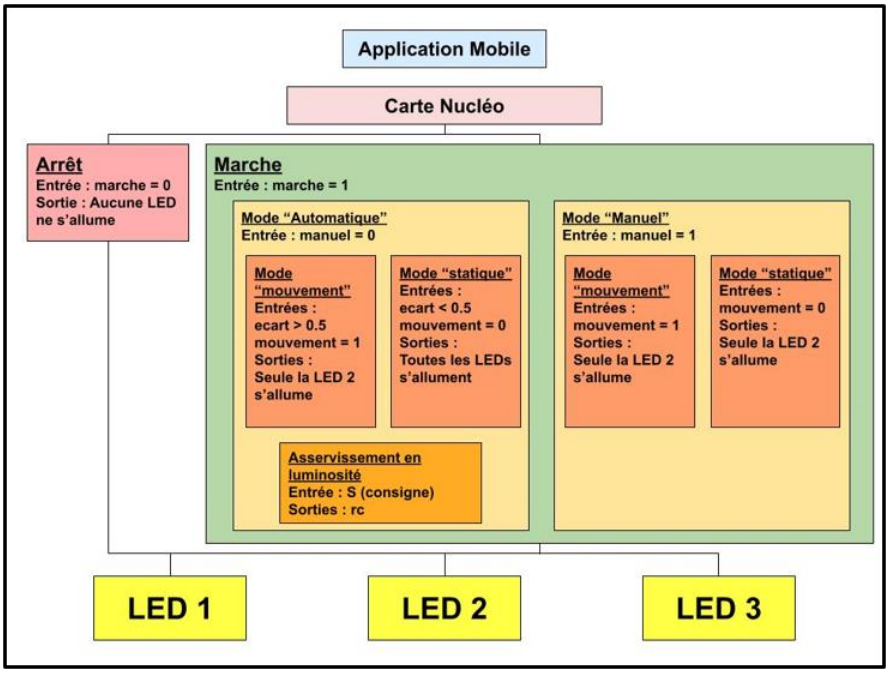

Figure n°9 : Schéma informatique du système

Notre télécommande tel que nous l'avons réalisée possède deux modes de fonctionnement différents : un mode automatique et un mode manuel.

a. *Mode manuel*

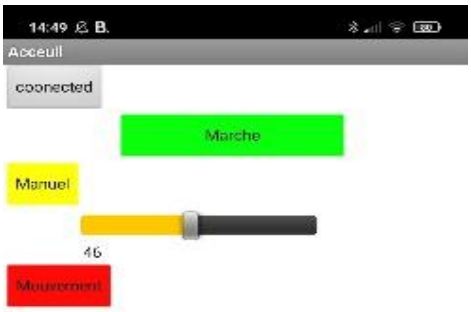

Figure 10 : Interface de l'application en mode Marche et Manuel

Le mode manuel est le mode le plus simple. Une fois la télécommande connectée en Bluetooth au module HC-05 on peut commander le fonctionnement ON/OFF sur le bouton *« Marche »*. L'intensité des LED se commande naïvement avec un curseur entre 0 (rapport cyclique de 0 donc pas de lumière) et 100 (rapport cyclique de 1 donc pleine puissance). Enfin le dernier paramètre que l'on peut contrôler est le mode mouvement ou le mode arrêt. Dans le mode mouvement, on considère que la vision est réduite champ directif donc une seule LED est allumée tandis que dans le mode arrêt les trois LEDs sont allumées.

Le code MBED du mode manuel est simple : on attribue à la variable « manuel » la valeur de la variable *« M »* de l'application, si elle vaut 1 alors le mode manuel est actif. On vérifie ensuite si la variable *« mouvement »* (qui correspond à la variable *« m »* de l'application) vaut 1 ou 0. Si elle vaut 1 alors on allume uniquement la LED centrale avec un rapport cyclique donné dans la variable *« rc »* qui correspond à la valeur du curseur de l'application (variable *« s »*).

b. ***Mode automatique***

1) Présentation

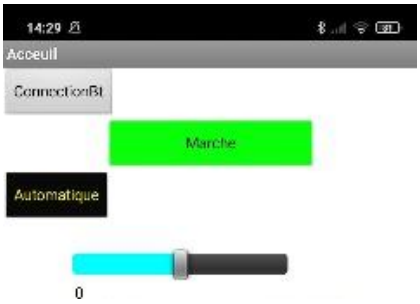

Figure 11 : Interface de l'application en mode Marche et Automatique

Le mode automatique est un mode ou on passe une intensité de consigne en commande uniquement (0 intensité nul et 100 intensité maximal arbitraire). L'intensité de sortie est ensuite asservie sur la valeur consigne. Enfin un accéléromètre est sensible au mouvement et réduit automatiquement le champ éclairé lors du déplacement du porteur de la lampe.

2) Asservissement

Nous avons commencé par essayer un asservissement de type proportionnel. Cependant n'ayant aucunes mesures absolues d'intensité nous ne pouvons pas nous rendre compte de ce qui se passe réellement, on ne peut pas se rendre compte de l'apparition d'écart statique et c'est difficile de définir un gain proportionnel. Pour pallier ce problème, on utilise un correcteur proportionnel intégrale (PI).

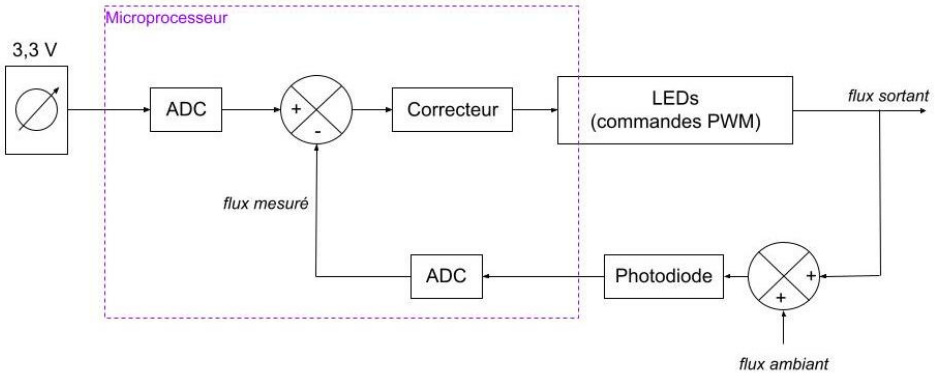

Figure n°12 : schéma bloc du système

3) Accéléromètre

Nous utilisons un accéléromètre pour estimer le mouvement de l'utilisateur. Pour cela, on a simplement branché l'accéléromètre sur une entrée analogique de la carte nucléo. Dans le code, on crée une fonction *« accelerometre »* qui permet de calculer l'écart-type des dix dernières valeurs reçues dans l'entrée analogique A0. Nous avons choisi de calculer l'écart-type car si ce dernier dépasse un certain seuil, cela signifie que l'accéléromètre détecte un mouvement. Le seuil choisi ici est 0,5 mais est assez arbitraire : le véritable seuil devra être établi après des tests en conditions réelles.

Dans la suite du code, si l'écart-type dépasse le seuil, on considère que nous sommes dans le mode « mouvement » et donc seule la LED centrale est allumée, et s'il est en dessous, on considère que nous sommes dans le mode « arrêt » et les trois LEDs sont allumées. Le rapport cyclique de chaque LED est déterminé par l'asservissement.

VI. Consommation de notre système

Dans le cahier des charges, il était demandé que notre système puisse avoir une autonomie de quelques heures.

A pleine luminosité sur notre application embarquée (avec les trois LEDs allumées), notre système nécessite un courant d'alimentation de 1,2A. Il faudrait donc une batterie externe de 12 A.h pour que la lampe frontale fonctionne 10h en première approximation.

Or, notre prototype a une alimentation de 4 A.h et délivre une tension d'alimentation de 7.2V (technologie NiMH), ce qui satisfait le cahier des charges. En revanche, celle-ci est beaucoup trop volumineuse et lourde et ne peut pas être utilisée pour une lampe frontale.

VII. Propositions d'amélioration

Le projet est actuellement au stade de prototype. De nombreuses améliorations peuvent être apportées.

D'abord, le système actuel n'est pas réellement portable : il est autonome et alimenté par une batterie externe mais on ne peut pas encore en faire un casque lumineux acceptable. On pourrait en effet utiliser une carte nucléo nano et une batterie moins imposante pour mieux miniaturiser le système. Pour la batterie, nous conseillons des batteries avec des technologies plus récentes (LiPo, Li-ion) qui offrent les mêmes performances pour un volume et un poids bien moindre.

Un autre problème que nous avons rencontré est que la LED, à grand ampérage, chauffe beaucoup. Pour éviter que les composants fondent, il faudrait rajouter un radiateur derrière les LEDs.

Enfin, durant la réalisation du prototype, nous n'avons pas pu tester le système en conditions réelles (environnement sombre et aucune lampe extérieure). De ce fait, nous avons réalisé un circuit électrique (expliqué plus tard) dans lequel nos lampes ne sont pas utilisées à leurs capacités maximales. La condition du cahier des charges d'avoir un flux de 500 lm n'est donc pas respectée dans l'état actuel du système. Cette condition devra être vérifiée pendant les tests sur le terrain.

VIII. Bilan du projet

Au cours de ce projet de groupe, nous avons pu mettre à profit nos compétences en électronique, en informatique et en automatique pour mettre au point un système LED asservi, qui peut servir de prototype pour une lampe frontale.

Nous avons appris à gérer un projet de groupe, en se répartissant les tâches pour obtenir un prototype final qui respecte le cahier des charges demandé par la société SOLEC, tout en restant flexible pour s'adapter au divers problèmes rencontrés (absences de membres du groupe, différences en termes de compétences en électronique). Par ailleurs, nous nous sommes efforcés de nous en tenir au planning prévisionnel mis au point en début de projet.

Si le cahier des charges est respecté, plusieurs pistes d'amélioration sont envisageables (autonomie, miniaturisation, puissance lumineuse).

IX. Code MBED

```
/* Projet PROTIS 2A Palaiseau 2022
Membres de l'équipe de développement : Aoustin Emma, Denis Angèle, Dupuis Tom, Lebegue Pie
rre
 */

// Déclaration des bibliothèques utiles
#include "mbed.h"
#include "platform/mbed_thread.h"
#define BLINKING_RATE_MS                                                    500

// Déclaration connexion PC
Serial pc(USBTX, USBRX, 9600); //liaison tera term
Serial bt(PA_9, PA_10, 9600);

// Déclaration des fonctions tickers
Ticker accelerometre_ticker;
Ticker luminosite_ticker;

// Déclaration des fonctions
void accelerometre(void);
void luminosite(void);
void rx_getData(void);

// Déclaration des variables PWM commandant les LEDs
PwmOut LED_pwm1(D5);
PwmOut LED_pwm2(D6);
PwmOut LED_pwm3(D9);

// Déclaration entrées du microprocesseur
AnalogIn acc(A1); // sortie de l'accéléromètre
AnalogIn photodiode(A2); // sortie de la photodiode
DigitalIn btn1(PC_13); //Bouton Bluetooth de la carte Nucléo

// Déclaration des variables de réceptions bluetooth
char datas[10]; // Chaine reçue
int indice_datar; // Indice de position dans la chaine


// Variables pour la mesure accéléromètre
double tab[10] = {0,0,0,0,0,0,0,0,0,0}; // Mesure glissante de la mesure de l'accéleromètr
e
double acceleration ;
double ecart=0;


// Variables pour la mesure photodiode
double iphoto;
double Tab[10] = {0,0,0,0,0,0,0,0,0,0}; // Mesure glissante de la mesure de la photodiode
double lum;// valeur mesuré sur la photodiode


// Déclaration des variables définies par l'application
int marche ;
int manuel ;
int mouvement ;
int jour;
double rc=0; // Rapport cyclique
```

```
double consigne = 0.2 ; // Valeur de consigne d'asservissement
double lum_max = 0.84; // Maximum de la mesure de la photodiode
double lum_min = 0.053; // Minimum de la mesure de la photodiode
double Kprop = 0.5 ;// Gain de correction proportionnel
double Kint = 0.01; // Gain de correction intégrale



int main() {
    bt.attach(&rx_getData); //connexion Bluetooth
    pc.printf("Connexion Bluetooth établie \r\n");
    accelerometre_ticker.attach(&accelerometre, 0.045); //mesure de l'accélération
    luminosite_ticker.attach(&luminosite, 0.01); //mesure de la luminosité

    //Initialisation des LEDs
    rc = 0;
    LED_pwm1.period_ms(2);
    LED_pwm2.period_ms(2);
    LED_pwm3.period_ms(2);


    while(1){
        // Lecture des données de l'application par bluetooth
        if (datas[0]=='b') { marche = datas[1]; }
        if (datas[0]=='M') { manuel = datas[1]; }
        if (datas[0]=='m') { mouvement = datas[1]; }
        if (datas[0]=='j') { jour = datas[1]; }
        if (datas[0]=='s') { rc = datas[1];
                             rc = rc/100;
                            }
        if (datas[0]=='c') {
            consigne = datas[1];
            consigne = consigne/100;
            consigne = consigne *( lum_max-lum_min) + lum_min;
        }


        //on vérifie que l'on a bien lancé l'application et appuyé sur le bouton "marche"
        if (marche==1) {

            // quand manuel = 0, on est en mode "automatique", on réalise donc un asservis
sement sur la luminosité et sur l'accélération
            if (manuel==0){
            if (ecart*10>0.5) mouvement = 1; //l'utilisateur bouge
            if (ecart*10<0.5) mouvement = 0; //l'utilisateur ne bouge pas
                if(mouvement ==1){
                    LED_pwm1.write(0);
                    LED_pwm2.write(rc);
                    LED_pwm3.write(0);
                }
                if(mouvement ==0){
                    LED_pwm1.write(rc);
                    LED_pwm2.write(rc);
                    LED_pwm3.write(rc);
                }
                // Asservissement proportionnel et intégral
                rc = Kint*rc + Kprop*(consigne - lum)/lum_max ;
                if (rc < 0){
                    rc = 0;
                    }
                if (rc > 1){
```

```
                    rc = 1;
                    }

            }
            // quand manuel = 1, on est en mode "manuel", l'utilisateur peut choisir s'il
est en mouvement ou non, et peut choisir la luminosité des LEDs
            if (manuel==1) {
                if (mouvement==1) {
                    LED_pwm1.write(0);
                    LED_pwm2.write(rc);
                    LED_pwm3.write(0);
                    }
                if (mouvement==0) {
                    LED_pwm1.write(rc);
                    LED_pwm2.write(rc);
                    LED_pwm3.write(rc);
                    }
                }
        }

        //si on appuie sur le bouton arrêt de l'application
        if (marche==0) {
            LED_pwm1.write(0);
            LED_pwm2.write(0);
            LED_pwm3.write(0);
        }



        }
}
```

```
void luminosite() {
/* Fonction d'interruption du ticker qui permet de mesurer l'intensité reçue par notre pho
todiode

La fonction permet de lire la valeur de luminosité ambiante mesurée par la photodiode, ell
e fait
une moyenne des 10 dernières valeurs reçues afin de ne pas prendre en compte les variation
s trop
brusques

ENTREES : pas d'entrées
SORTIE : lum - valeur de la luminosité de la photodiode moyennée sur les 10 dernières vale
urs mesurées

*/
    int i ; double S;
    S = 0;

    for (i = 0; i < 9; i++){
        Tab[i]= Tab[i+1];
        S = S + Tab[i];
        }
    Tab[9]= photodiode.read(); // on récupère la valeur lue par la photodiode et on la con
vertit en une valeur entre 0.0 et 1.0
    lum = (S+Tab[9])/10;
}
```

```
void accelerometre() {
/* Fonction d'interruption du ticker qui permet de mesurer la valeur de l'accélération

La fonction permet de lire la valeur mesurée par l'accéléromètre, elle calcule un écart
type sur les 10 dernières valeurs mesurées afin de prendre en compte les variations
faibles de l'accélération

ENTREES : pas d'entrées
SORTIE : ecart - renvoie la valeur de l'écart type des 10 dernières valeurs mesurées

*/
    int i; double somme; double value;
    somme = 0;
    for (i = 0; i < 9; i++){
        tab[i]= tab[i+1];
        somme = somme + tab[i];
    }
    tab[9] = acc.read(); // on récupère la valeur lue par l'accéléromètre et on la convert
it en une valeur entre 0.0 et 1.0
    value = (somme + tab[9])/10;
    ecart = sqrt(0.1*((tab[0]-value)*(tab[0]-value)+(tab[1]-value)*(tab[1]-value)+(tab[2]-
value)*(tab[2]-value)+(tab[3]-value)*(tab[3]-value)+(tab[4]-value)*(tab[4]-value)+(tab[5]-
value)*(tab[5]-value)+(tab[6]-value)*(tab[6]-value)+(tab[7]-value)*(tab[7]-value)+(tab[8]-
value)*(tab[8]-value)+(tab[9]-value)*(tab[9]-value)));
    }



void rx_getData() {
/* Fonction d'interruption du ticker qui permet de récupérer les donnnées de l'application
 Bluetooth

ENTREES : pas d'entrées
SORTIE : création d'une liste datas contenant les variables définies sur l'application

*/

    char c = bt.getc();
    pc.putc(c);
    if(c == 255){
        indice_datar = 0;
    }
    else{

        datas[indice_datar] = c;
        indice_datar++;
    }
    if(indice_datar == 2){
            pc.putc(datas[0]);
            pc.printf("%d\r\n",datas[1]);
            indice_datar++;
            }
}
```