Louis ETIEN
Hugo HAJAALI
Aurchi RAHMAN
Lorenzo SARGENI

Compte-rendu final - Spectroscope

Institut d'Optique Graduate School

Nous attestons que ce travail est original, que nous citons en référence toutes les sources utilisées et qu'il ne comporte pas de plagiat.

L'objectif de ce projet est de réaliser un spectromètre. On dispose pour cela d'une lampe à vapeur de mercure, d'un goniomètre sur lequel on place un réseau de diffraction en transmission, d'une carte nucléo, et d'un capteur lumineux de référence TSL201R-LF, une barrette composée de 64 pixels lumineux.

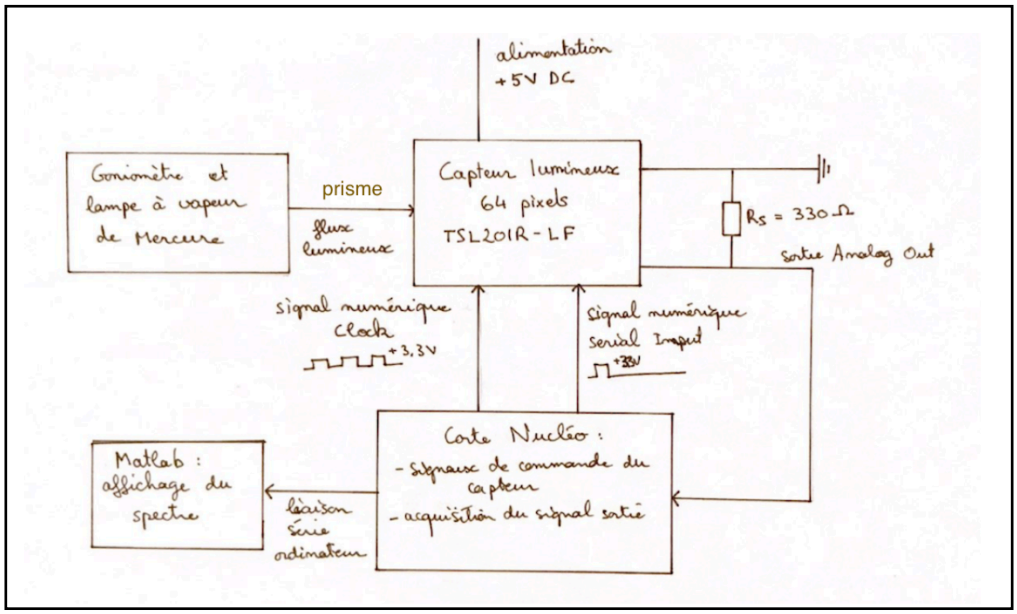


Fig 1. Schéma fonctionnel du spectromètre

I. Diffraction et focalisation de la source lumineuse sur le capteur

Nous avons mesuré la distance focale en utilisant la conjugaison infini-foyer et pour cela nous avons placé une mire graduée au foyer de l'objectif qui sert d'objet à l'infini. La distance focale a été obtenue à l'aide de la mesure de la dimension de l'image agrandie de par l'objectif de microscope sur le réticule de l'oculaire. Nous avons mesuré la dimension angulaire α du collimateur. Cette valeur est ensuite divisée par 20, qui correspond au nombre de graduations du collimateur, pour connaître la dimension angulaire pour chaque graduation. Ainsi pour obtenir la dimension angulaire de la mire, pour des positions différentes le long du banc, on mesure le nombre de graduation du collimateur correspondant au nombre de graduation du microscope et on obtient par la relation . Le grandissement de l'objectif du microscope nous sert à obtenir après la mesure de to qui sont reliés par la formule .

AN: on obtient f'= 173 mm

Pour faire parvenir les raies du premier ordre de la lampe à mercure jusqu'à la barrette CCD, nous avons utilisé un goniomètre dans lequel nous avons placé un objectif convergent et une fente réglable située dans le plan focal. La fente est éclairée et donne une image à l'infini qui joue le rôle d'un objet à l'infini. Le goniomètre, à partir de la source lumineuse (lampe à mercure), diffracte la lumière à l'aide d'un prisme/réseau à

un angle donné, qui est ensuite récupéré par la barrette CCD positionnée selon l'angle auquel la lumière est diffractée. Les interférences qui ont lieu entre les faisceaux diffractés par le réseau à des angles différents dépendent de la longueurs d'onde des faisceaux considérées. Ainsi on obtient des pics de la lumière caractéristique de l'angle d.

La barrette est positionnée dans une tube à la distance focale approximative trouvée précédemment. En plaçant une feuille selon l'angle de diffraction de la lumière incident, nous observons des franges très fines parallèles au réseau. Les franges sont positionnées selon leurs longueurs d'onde, la déviation des faisceaux est d'autant plus grande que les longueurs d'ondes sont grandes. On obtient un motif répétitif et les franges correspondent à des ordres différents du spectre diffracté selon leur positionnement (la frange centrale est de l'ordre k=0). Avec les faisceaux récolté sur la barrette CCD, nous étudions l'intensité des différentes longueurs d'ondes du spectre de la lampe à mercure.

Nous avons relié la barrette CCD à un breadboard, et nous avons utilisé une résistance de 330 ohm. Premièrement, nous avions relié le breadboard avec deux générateurs de signaux, une alimentation de tension et l'oscilloscope pour vérifier le bon fonctionnement de la barrette et pour visualiser le signal obtenu.Ce montage est remplacé par la carte nucleo qui sera relié avec le breadboard. La carte nucleo nous aide à transformer le signal numérique en signal analogique qui va ensuite nous fournir des courbes d'intensité de chaque longueur d'onde diffracté du spectre de la lampe de Mercure.

II. Fonctionnement de la barrette

La barrette est composée de 64 pixels régulièrement espacés. Outre l'alimentation et la masse, la barrette comporte 3 autres broches importantes: le signal de sortie AO, le signal de commande de type horloge CLK, et le signal de commande de type impulsion SI.
Le signal de commande CLK est un signal créneau, dont la période détermine la vitesse de fonctionnement de la barrette. Entre deux fronts montants du signal créneau, le signal de sortie AO renvoie la valeur du pixel en cours de lecture, et chaque front montant fait passer AO au pixel suivant.
Il est conseillé que les signaux de commande fonctionnent avec une tension de 0 à 5V (0V = 0 et 5V =1 en langage logique).
En absence de signal SI, la sortie AO est nulle par défaut. Pour démarrer la lecture des pixels, la barrette doit recevoir un signal d'impulsion au bon moment. Le signal SI doit être à 1 lors d'un front montant de l'horloge, et descendre à 0 avant le prochain front montant.

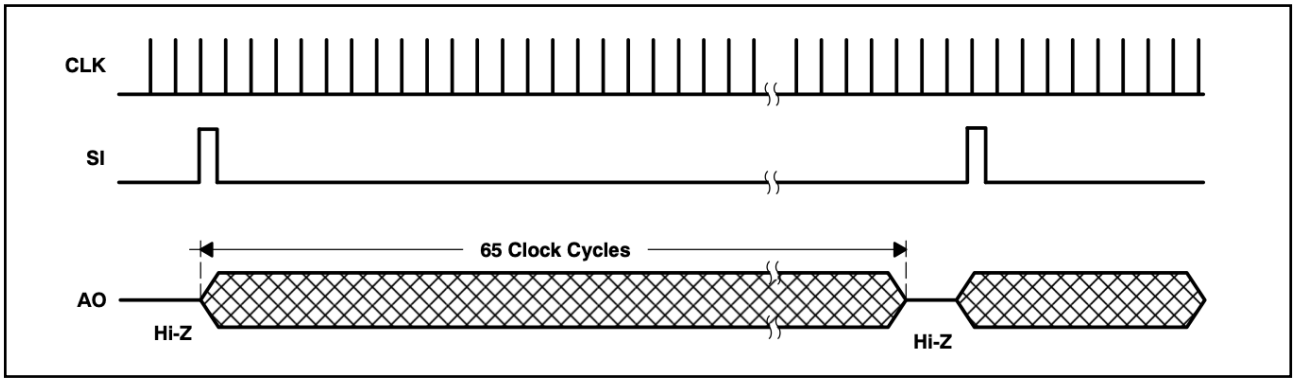


Fig 2. Positionnement de l'impulsion [1]

Chaque pixel consiste en une photodiode reliée à un intégrateur. Pendant le temps d'intégration, la sortie de l'intégrateur est reliée à un condensateur de charge. Pendant la lecture du pixel, le condensateur de charge est relié à la sortie AO et le condensateur de l'intégrateur est déchargé.

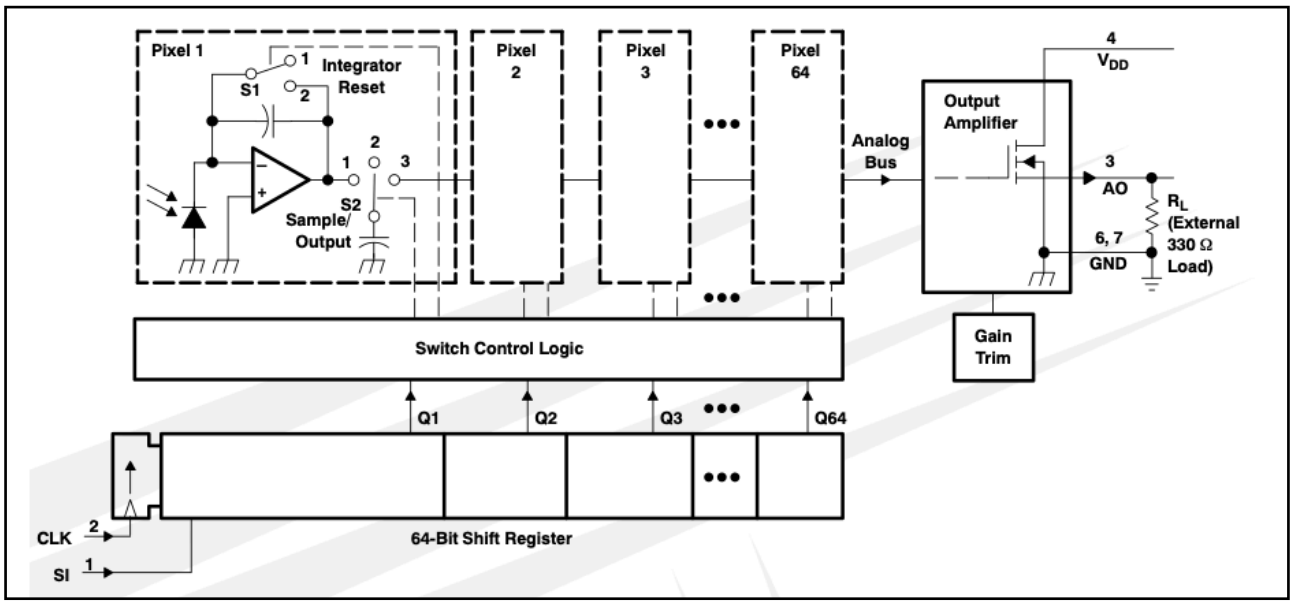


Fig 3. Schéma électrique du capteur lumineux [1]

La commutation des interrupteurs S1 et S2 (voir schémas) est assuré par un commutateur 64 bits. Une fois l'acquisition lancée avec le signal SI, le commutateur passe d'un pixel à l'autre à chaque front montant de l'horloge, et se remet à zéro au 64e pixel.
Le temps d'intégration correspond au temps qui s'écoule entre deux appels d'un pixels, soit la période du signal SI moins la période du signal horloge.

Montrons que, suite à une période d'intégration, la charge Q_c du condensateur de charge est proportionnelle au courant de la diode i_d et au temps d'intégration τ_i:

On considère que l'Amplificateur Opérationnel est idéal et fonctionne en régime linéaire.

On a alors : $i_+ \simeq 0,\ i_- \simeq 0$, et $V_+ \simeq V_- = 0$.

Le courant de la diode passe donc dans le condensateur de l'intégrateur, et on a, avec u la tension du condensateur : $i_d = C\frac{du}{dt}$, soit en intégrant sur le temps d'intégration : $u = \frac{i_d \tau_i}{C_i}$

Au niveau du condensateur de charge, on aura donc : $Q_c = C_c u(\tau_i) = \frac{C_c}{Ci} i_d \tau_i$

Le courant de la diode est proportionnel à l'éclairement, la charge du condensateur de charge est donc bien proportionnelle à l'éclairement et au temps d'intégration.
Pour un fort éclairement, la barrette peut donc saturer: les condensateur de charge atteignent leur charge maximale avant la fin de l'intégration. Pour y remédier, on peut réduire le temps d'intégration en réduisant la période de l'horloge. La fréquence maximale à laquelle la barrette peut fonctionner est 5000 Khz, bien supérieure à la limite d'une carte nucléo, 20 Khz.

Remarque: il est d'usage de faire tourner quelques cycles d'acquisition « à vide » avant de lire le signal envoyé par barrette. En effet, les condensateur se chargent par défaut, dès que la barrette est alimentée, même en l'absence d'impulsion. A la première impulsion reçue par le capteur, les condensateurs sont donc tous saturés, et la tension renvoyée par AO ne reflètera pas l'éclairement reçu par la diode.

III. Programmation de l'acquisition

Voici les étapes principales de la programmation de l'acquisition :

- Générer un signal clock PWM (=Pulsed Width Modulation) sur lequel s'appuie tout le reste du programme pour lancer le signal impulsion au bon moment et afin de compter en temps réel le nombre de pixels dont le signal a été acquis
- Chaque front descendant déclenche une fonction d'interruption qui enregistre la tension de chaque pixel dans un tableau et comptabilise le nombre d'acquisition et de séries d'acquisition effectuées. Les descentes de front du signal clock sont les évènements élémentaires sur lequel tout le programme s'appuie
- Une fois une série achevée, une boucle for se charge de parcourir le tableau de données et les transmets à Matlab via la liaison série établie

Voici les caractéristiques techniques :

- Le signal clock a une fréquence de 20kHz, ce qui correspond à une période de 50 microsecondes. Pour obtenir un créneau, nous avons choisi un rapport cyclique de 50%
- Un objet InterruptIn fait la veille du signal clock et déclenche la fonction d'interruption à chaque front descendant. La tension est sauvegardé dans un tableau d'entier, plus particulièrement des unsigned short int, qui sont des entiers codés sur 16 bits. Nous avons fait ce choix puisque la tension relevée est codée sur 16 bits. On minimise dans ce cas la mémoire.
- Chaque passage dans la fonction d'interruption est comptée, en utilisant une variable dédiée *compteur*. Lorsque 100 fronts descendants consécutifs ont été observés, une impulsion est déclenchée qui ne s'interrompera qu'une fois le 101è front descendant observé : c'est l'impulsion qui déclenche une nouvelle acquisition des pixels de la barette.
- Afin d'obtenir des mesures de qualité, nous ne transmettons les données qu'à partir de la 6ème acquisition. Les premières ne sont en effet pas fiables puisque les pixels du capteurs sont saturés.
- Une fois qu'une acquisition est terminée, les 100 valeurs stockées au sein de notre tableau d'entiers sont envoyées via la liaison série vers l'ordinateur.
- Matlab les recevra et les traitera alors afin que nous puissions les exploiter, c'est-à-dire en tracer le graphe pour observer le spectre de notre source lumineuse.

IV. Transmission de la carte Nucléo vers Matlab

La transmission des données collectées par la carte Nucleo vers Matlab se font grâce à une liaison série et le protocole RS232.
C'est un protocole asynchrone, c'est à dire qu'il n'y a pas de signal horloge pour synchroniser l'émetteur et le récepteur. Ce protocole de transfert de données est dit point à point, c'est à dire qu'il n'y a que deux noeuds qui peuvent discuter ensemble. Les échanges se font en Full-Duplex, autrement dit, il y a deux supports physiques entre les deux noeuds. Les données peuvent aller dans les deux sens en même temps. Les deux signaux distincts sont RX (réception) et TX (transmission). Cette liaison se fait dans notre cas par un cable USB.

Pour initialiser cette liaison il faut mettre « Serial NOM_LIASON(USBTX, USBRX); » au début du projet sur Mbed, pour déclarer quelles broches sont utilisées pour la liaison. Dans notre cas la vitesse de transmission est de 9600 bauds, soit pour une liaison RS232 9600 bits/secondes. Nous n'augmentons pas la vitesse de transmission car la carte Nucleo est déjà limitante pour la partie optique avant cette limite de 9600 bauds.

D'un point de vue plus technique, pour transmettre des données il faut utiliser la commande « pc.printf » avec comme argument les données à transmettre. Un aspect très important est de configurer le *terminator,* c'est à dire quels est le marqueur de séparation entre deux valeurs différentes. Il faut spécifier cela à Matlab et sur la carte Nucleo. Dans

notre cas c'est le duo CR/LF, retour au début de la ligne, carriage return (CR), \r et LF, Line feed, saut de ligne, \n.
Les données sont transmise comme ceci par la Nucleo « pc.printf("%d \r\n", valeur[i]); ».
Matlab reçoit donc les valeurs comme-ceci (Fig. 3)

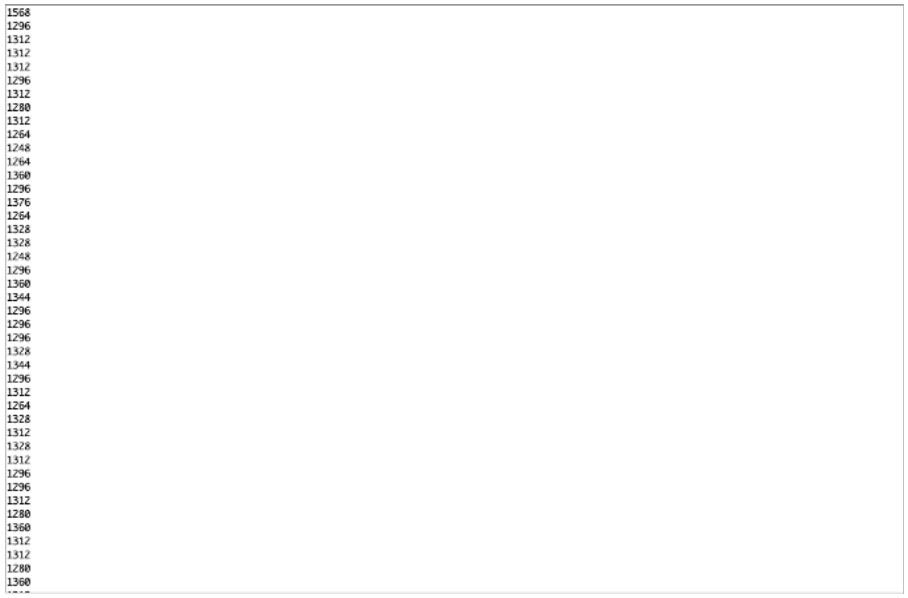


Fig 4. Réception des données par Matlab

Dans cet exemple, le capteur était dans une pièce relativement sombre car rappelons le, les valeurs des pixels prisent par la carte Nucleo sont sur 16 bits, soit 65536 valeurs possible.
Du côté de Matlab, Matlab va scruter en permanence la liaison série (initialisé préalablement) et dès qu'il va détecter un *terminator* il va enregistrer la donnée le suivant. Ce scrutage se fait par le biais de la fonction « configureCallback » qui appelle la fonction « extraction » qui récupère la donnée (cf code). Dès qu'elle aura récupérés les 64 valeurs des pixels le callback sera désactiver et cessera donc l'acquisition. Nous mettons ensuite en forme les données sous formes de graphes. Une fois la fonction lancée on a 10 secondes pour récupérer les données.

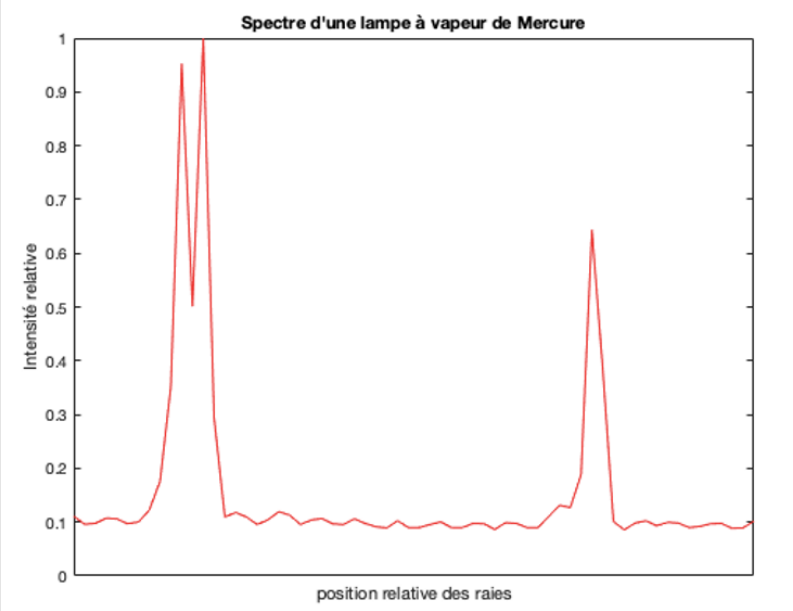
La fonction et le script d'initialisation de la liaison Matlab/Nucleo se trouvent en annexe.

V. Tests et validation

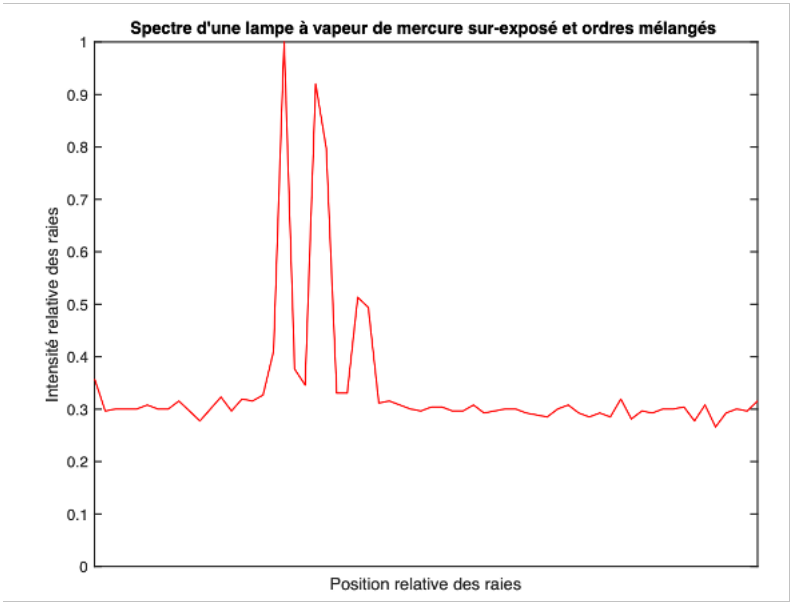
La première étape du projet consistait à faire fonctionner le capteur lumineux avec des signaux de commande générés par des GBF. On a pu tester le fonctionnement du capteur simplement en connectant sa sortie à un oscilloscope.
L'étape suivante fut de générer ces signaux de commande à l'aide de la carte nucléo. Avant de connecter la carte nucléo au capteur lumineux, on s'est assuré que les signaux générés étaient bien ceux souhaités en observant les sorties de la carte à l'aide d'un oscilloscope. On a pu observer que lorsque l'on se rapproche de la fréquence limite de fonctionnement de la carte nucléo, le signal présente des pics d'oscillation au niveau des fronts montants et descendants (dûs à la décomposition en série de Fourrier du signal créneau), mais dont la hauteur, inférieure à 2.5 V, ne suffit pas à perturber le fonctionnement du capteur.
La dernière étape fut de tester la transmission en liaison depuis la carte nucléo vers l'ordinateur. Pour s'assurer que transmettre des données avec la carte nucléo tout en effectuant des actions dans une fonction d'interruption, nous avons écrit un programme transmettant des données teste, et observé les sorties de la carte nucléo sur l'oscilloscope.
La dernière étape du projet était l'observation des raies du mercure.

VI. Résultas finaux

Le résultat finale est ci-dessous :



Un autre exemple moins concluant est le suivant :



On observe par exemple par la présence d'une raie en plus par rapport au doublet jaune du mercure classique. Cela veut donc dire que l'on a superposé plusieurs ordres. Cela est du au mauvais réglage et positionnement du prisme diffractant. On voit aussi qu'il y a un offset assez haut. Cela est du à la lumière parasite de la lampe à vapeur mercure elle même. C'est pour cela qu'il faut bien penser à mettre le tube contenant le capteur à 90° par rapport à la source lumineuse, pour éviter au plus de la lumière parasite.

V. Annexe

Code en C sur carte Nucléo :

```
#include "mbed.h"


#define Tclock 0.00005 // 20 kHz


Serial pc(USBTX, USBRX); //configuration de la liason série
PwmOut clk(D9);
DigitalOut Imp(D3,0);
InterruptIn cpt(D6);  // a relier à la broche clk sur le capteur ( qui est aussi relier à D9 ),
// Permet de compter le nombre de clock pour lancer l'impulsion et donc commencer
l'acquisition des données.
AnalogIn analog_in(A0); // lit les données en sortie du capteur ( relier la broche A0 de la
barette à A0 du nucleo )

int nacq = 0; // compteur pour eviter la saturation du à la surexpostion du premier jeu de
données
int compteur = 0;  // permet de compter le nombre de clock pour lancer l'impulsion
unsigned short int meas_16 = 0; // A0 sera stocké ici, taille de 16 bit pour tout ordinateur
unsigned short int  valeur[100]; // stock les données

void inter(void);


int main(){

   clk.period(Tclock);
   clk.pulsewidth(Tclock/2);
   cpt.fall(&inter);

   while(1){
      if(nacq == 5 ) {
         for (int i = 0 ; i < 100; i++) {
           pc.printf("%d \r\n", valeur[i] ); // \r\n se réferer à la transmission vers matlab, \r
retour à la ligne sans saut

         }
      }
   }
```

```
}

void inter(){
   compteur = compteur + 1;
   if (compteur == 100) {
      compteur = 0;
      Imp = 1;
      nacq++;
   }
   if(compteur == 1) {
      Imp = 0;
   }
   meas_16 = analog_in.read_u16() ;
   valeur[compteur]=  meas_16 ;
}
```

Code Matlab :

```
nucleo = serialport(« /dev/cu.usbmodem14203",9600) % permet de configurer la liaison série, le % nom est propre à chaque PC, 9600 est la vitesse de transmission.

configureTerminator(nucleo,"CR/LF");
% chaque valeurs sera séparé par un CR ( retour au début de la ligne, carriage return, \r)
% et un LF (  Line feed, saut de ligne, \n), il faut bien penser à être
% en accord avec le code sur le micro contrôleur!

nucleo.UserData = struct("valeur",[],"Count",1)
% crée un objet "interne" à la liaison série qui pourra être appelé
% comme ceci "nucleo.UserData.valeur " ou  "nucleo.UserData.Count",
% l'utilisation de cet objet permet de stocker les données "globalement" et
% permet de mémoriser le nombre de donnée sauvegardés
% la variable « valeur » stockera les données transmise par le micro-contrôleur
% tandis que count sera un compteur pour dénombrer le nombre de mesures prisent
% "valeur" est un tableau et "Count" un réel

configureCallback(nucleo,"terminator", @extraction);
% appelle la fonction "extraction" dès qu'un "terminator" est détectée, c'est à dire
% dès que la combinaison CR\LF est detectée ( "\r\n" ) => lit une donnée à la fois ! Elle sera
% donc appelée 64 fois (nombre de photodiode sur le capteur )

%% Pour plus détails se referer à :
```

```
%  https://fr.mathworks.com/help/matlab/import_export/read-streaming-data-from-arduino.html



function extraction(device,~ )
%device est le nom de la liaison série, ici nucleo
data = readline(device) ; % lit une donnée à la fois

device.UserData.valeur(end+1) = str2double(data);
% conversion d’un type 'string' à une valeurs numérique

device.UserData.Count = device.UserData.Count + 1;
% permet de compter le nombre de valeurs enregistrer

if device.UserData.Count > 65
   configureCallback(device, "off");
   val = device.UserData.valeur(2:end) ;

   val = val/max(val) ;
   figure()
   plot(val)
   xlim([2,64])
   ylim([0,1])
   set(gca, 'XTick', []); % permet d’enlever l'axe des x


end
end
```

Références :

[1] documentation du composant TSL201-R-LF:
https://datasheet.octopart.com/TSL201R-LF-TAOS-datasheet-8327114.pdf