Projet IÉTI - Promo 2023

SPECTROMÈTRE

Rapport Technique

Table des matières

Introduction

Ce projet consiste à élaborer un spectromètre capable de fournir des courbes sur une machine (nous utiliserons ici des ordinateurs sous Windows). Il s'agit donc d'obtenir l'intensité relative d'une source lumineuse numériquement en fonction de la longueur d'onde. Il existe évidemment bien d'autres manières d'obtenir un spectre d'une source lumineuse. Cependant le faire numériquement présente de nombreux avantages : en effet, l'acquisition de ces données numériques permet de réaliser des traitements informatiques qui ne sont donc pas possible lorsque l'on se contente de diffracter une source sur un écran blanc.

En l'occurrence, ce spectromètre permet actuellement deux types d'acquisition. Une acquisition simple qui permet notamment de bien observer une raie spécifique. Et une acquisition étendue qui permet via plusieurs mesures d'observer le spectre de la source sur une large étendue spectrale.

Bien que le prototype soit fonctionnel, il existe encore bien des manières de l'optimiser, nous en discuterons à la fin de ce rapport.

Découpage fonctionnel

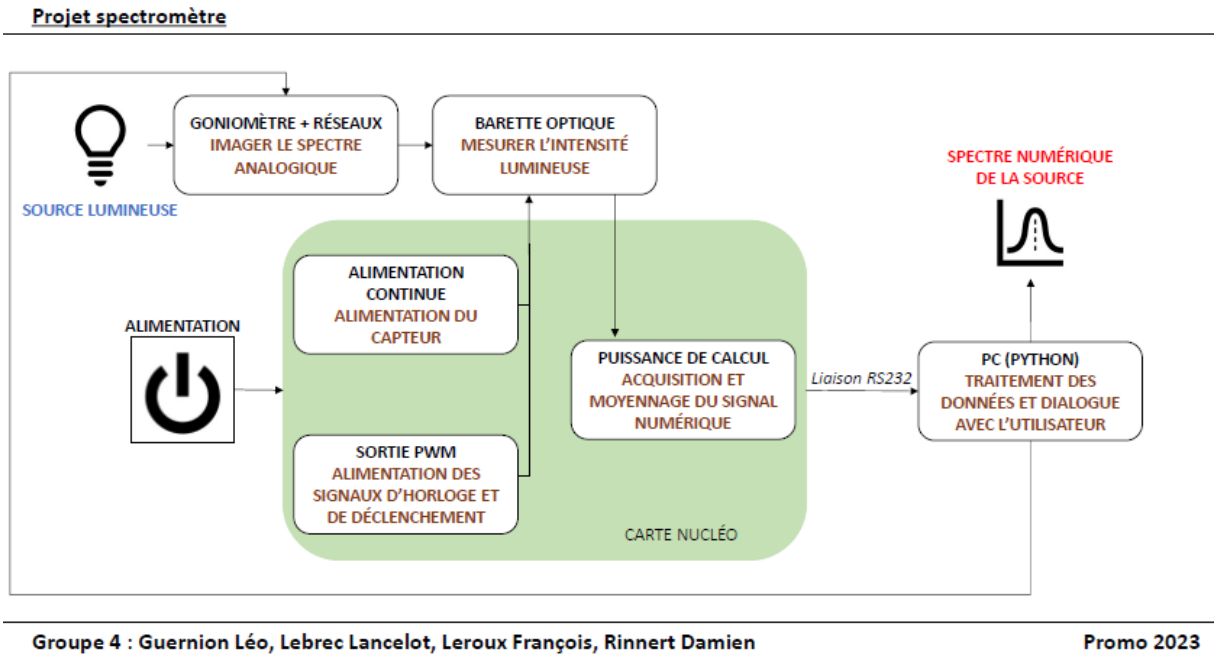


Figure 1 : Schéma bloc du projet

Afin de mener à bien le projet, différentes fonctionnalités ont dues être implémentées :

- Imager le spectre analogique : Il s'agit de diffracter le spectre de la source lumineuse, de sorte à ce que l'on puisse différencier les intensités correspondants à des longueurs d'onde différente en fonction de la position dans l'espace. Pour ce qui est de diffracter la source lumineuse nous utiliserons un réseau, et pour ce qui est de se repérer dans l'espace nous utiliserons un goniomètre.
- Mesurer l'intensité lumineuse : Il s'agit d'acquérir les données de l'intensité en fonction de l'espace. Pour cela nous utiliserons une barrette optique TSL201R du fabricant TAOS que nous fixons sur le goniomètre.
- Alimentation du capteur des signaux d'horloge et de déclenchement, acquisition et moyennage du signal numérique : Il s'agit d'alimenter le capteur de sorte à obtenir les valeurs de l'intensité pour les différents pixels qui le compose, en prêtant attention à ne pas saturer les signaux et à assurer une bonne fréquence de restitution. Pour cela nous utiliserons une carte Nucléo L476RG du fabricant STMicroelectronics.
- Communication Nucléo - Machine : Il s'agit de transférer les valeurs de l'intensité mesurées de la carte Nucléo vers l'ordinateur où nous utilisons Python pour le traitement des données. Afin d'assurer la communication entre les deux entités, nous utilisons une liaison RS232.
- Traitement des données et interaction avec l'utilisateur : Il s'agit d'appliquer les corrections voulues sur le signal mesuré et transféré, et de le mettre en forme afin d'obtenir les résultats sous forme d'un graphique. Pour se faire nous réaliserons les codes sur Python, ce qui nous permet également de communiquer avec l'utilisateur (via une interface graphique et la console Python) afin de lui fournir les indications pour l'étalonnage par exemple, ou encore les indications relatives aux acquisitions étendues.

Réaliser le prototype

Dispersion de la source, montage optique

Par Léo Guernion et Lancelot Lebrec

La partie optique du spectromètre s'appuie sur un goniomètre et un réseau comme présenté sur le schéma de principe suivant. Le réseau est fixe par rapport au goniomètre. L'ensemble {capteur, lentille convergente} est en rotation par rapport au goniomètre autour du réseau. Un résultat d'optique remarquable pour les réseaux est le suivant : $\sin(\theta_m) = \sin(\theta_i) - m\frac{\lambda}{d}$. Il indique pour chaque ordre du réseau m la direction suivi par le faisceau lumineux de longueur d'onde λ.

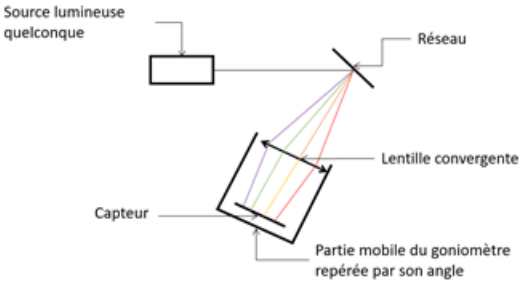


Figure 2 : Schéma de principe de la dispersion des faisceaux lumineux de la source.

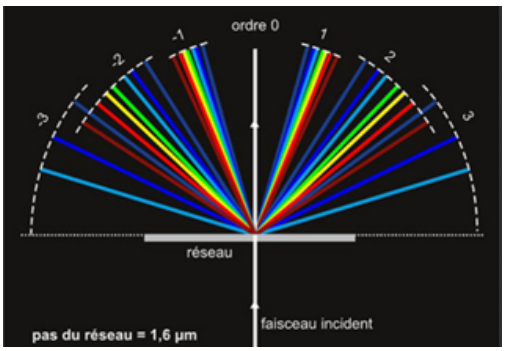


Figure 3 : Représentation des différents ordres de diffraction d'une source blanche par un réseau de pas 1,6 µm.

Comme indiqué ci-dessus, il y a plusieurs ordres de diffraction du réseau qui se répètent et ils contiennent tous l'ensemble du spectre de la source. Nous nous intéressons uniquement à l'ordre 1 car c'est le plus intense : il permet une meilleure détection.

Un réseau est caractérisé par son pas d mesuré en nombre de fentes par millimètre. Plus le pas est faible, plus les différents ordres se rapprochent les uns des autres ; ils finissent même par se recouvrir pour des pas trop petits. La principale contrainte pour le choix du réseau est donc que les ordres 1 et 2 ne se recouvrent pas. Nous avons fait différents essais et le réseau retenu a un pas de 600 traits/mm.

Ainsi nous parvenons à imager le spectre de la source dans le foyer image de la lentille (les faisceaux lumineux en sortie du réseau étant localisés à l'infini). C'est donc dans le foyer image de la lentille que nous devons placer la barrette optique.

Alimentation et acquisition, carte Nucléo

Par François Leroux

Une fois que l'on a réussi à focaliser une figure de diffraction, il nous faut mesurer l'intensité des différentes raies qui la composent. Pour cela, on utilise le capteur TSL201R de chez Taos, dont la documentation est donnée en **Annexe 1**. Ce capteur est une barrette de 64 photodiodes disposées sur la même ligne. En le plaçant dans le plan focal de la lentille qui focalise notre figure de diffraction, on peut obtenir l'intensité lumineuse de 64 points différents. En déplaçant plusieurs fois le capteur de manière à couvrir l'entièreté de la plage angulaire occupée par la figure de diffraction, on peut ainsi reconstituer la totalité du spectre de la source étudiée.

Le capteur est alimenté par la carte nucléo par trois tensions différentes : une tension d'alimentation continue de 5V (V_{DD}) et deux signaux créneaux : un signal de déclenchement (SI) permettant d'initialiser la restitution des mesures et un signal d'horloge (CLK) qui définit la fréquence de cette restitution.

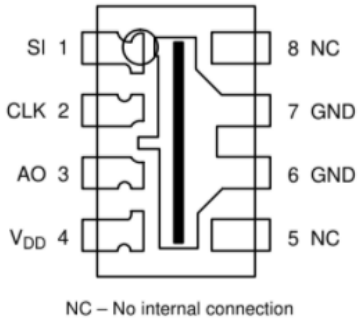


Figure 4 : Schéma descriptif du capteur TSL201R (extrait de la documentation)

Concrètement, après avoir reçu un front montant du signal de déclenchement, le capteur délivre sur la sortie A0 et à chaque front montant du signal d'horloge une tension proportionnelle au flux lumineux reçu par un de ses 64 pixels. Le $n^{ième}$ front montant du signal d'horloge déclenche la restitution de la tension proportionnelle à l'intensité lumineuse reçue par le $n^{ième}$ pixel. Après avoir enregistré 65 fronts montants du signal d'horloge, le capteur s'arrête de fournir une tension en sortie et se met en attente du prochain front montant du signal de déclenchement.

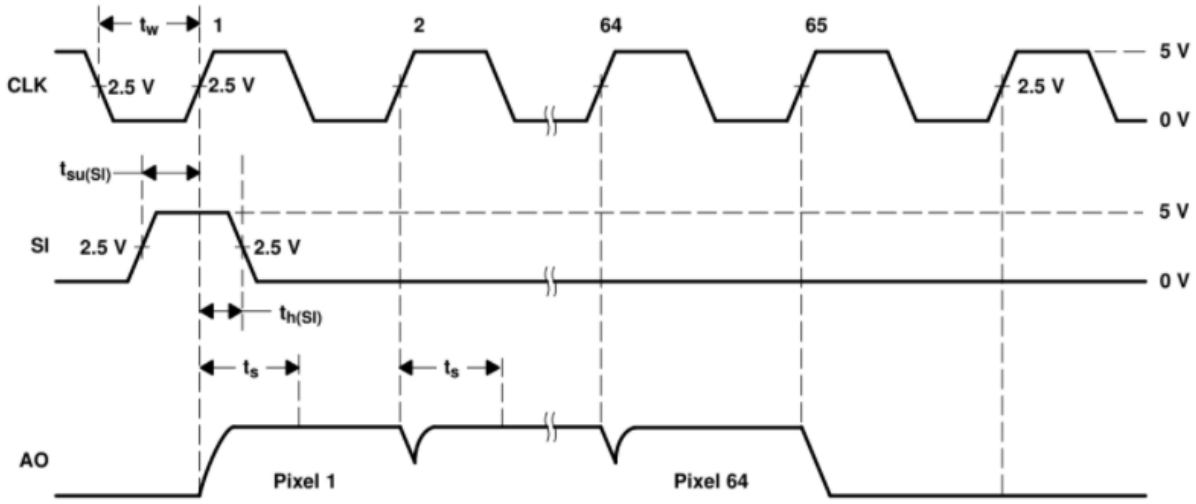


Figure 5 : Représentation des signaux de déclenchement, d'horloge et de sortie du capteur (extrait de la documentation)

Les 64 valeurs obtenues en sortie du capteur seront récupérées et moyennées via le Convertisseur Analogique Numérique (CAN) de la carte nucléo et transmises à l'ordinateur pour être post-traitées.

Rentrons à présent dans les détails en décrivant numériquement nos signaux et en expliquant la structure de notre code. La tension d'alimentation de la carte est fournie directement par le port 5V de la carte Nucléo. La documentation technique précisant que la carte détecte un front montant à partir du moment où la tension en entrée dépasse 2.5V, on va utiliser directement les sorties PWM de la carte, fournissant des signaux créneaux dont le temps haut correspond à 3.3V, pour générer les signaux de déclenchement et d'horloge.

La documentation technique du capteur impose une fréquence du signal d'horloge f_{clock} comprise entre 5 kHz et 5 MHz. Pour régler ce paramètre, il faut également tenir compte des limitations de la carte nucléo qui prend un certain temps à lire une valeur de tension. Cependant, avoir f_{clock} grand permet de réduire le temps d'intégration des photodiodes du capteur et ainsi de se prémunir contre une éventuelle saturation. En tenant compte de ces différents phénomènes, on a choisi f_{clock} = 20 kHz.

La période du signal de déclenchement doit être supérieure à 65 fois la période du signal d'horloge, afin de laisser le temps nécessaire à la restitution des 64 valeurs et à la valeur de contrôle imposé par le capteur. Pour avoir de la marge on a choisi $T_{SI} = 100 \times T_{Clock}$.

Par tâtonnement on constate qu'il est bon de régler le temps haut du signal de déclenchement de manière à ce qu'il soit deux fois supérieur à celui du signal d'horloge. Conformément à la documentation technique on prend 0.5 comme rapport cyclique du signal d'horloge. On en déduit celui du signal de déclenchement : $\tau = \frac{2\times 0.5\times T_{Clock}}{T_{SI}} = \frac{T_{Clock}}{100\times T_{Clock}} = 0.01.$

Le code de la carte nucléo est conçu de manière à pouvoir renvoyer à l'ordinateur un tableau de 64 valeurs correspondant à l'intensité relative reçue par chacun des pixels du capteur moyenné sur N mesures, N étant une constante symbolique du programme fixée à 100 dans notre cas.

L'acquisition de ces N mesures se fait à l'aide de la fonction *read()*, en connectant la sortie A0 du capteur sur une entrée analogique de la carte Nucléo. Afin de synchroniser l'acquisition avec la restitution des valeurs par le capteur on procède de la manière suivante : on connecte les sorties PWM de la carte nucléo correspondant aux signaux SI et CLOCK sur des entrées analogiques de la

carte. Ainsi, on peut désormais programmer une interruption sur un front montant du signal de déclenchement. Lorsqu'un tel événement advient, on exécute une série de 64 lectures de la tension de sortie de la carte nucléo que l'on ajoute respectivement aux 64 cases d'un tableau de double. Ces 64 lectures sont synchronisées avec le signal CLOCK via le branchement décrit juste au-dessus. Notons qu'une variable de contrôle, nommée *test* dans le code, permet de s'assurer de ne pas mesurer deux fois la même valeur. Un test est réalisé à chaque interruption afin de s'arrêter au bout de N itérations. Alors, on divise chaque élément de ce tableau de doubles afin de calculer la moyenne et on peut envoyer l'information à l'ordinateur.
Les codes relatifs à la carte Nucléo se trouvent en ***Annexe 2***

Réalisation de l'interface graphique sous Python

Par Damien Rinnert

L'interface graphique permet à l'utilisateur d'interagir avec les fonctions Python appropriées en cliquant sur des boutons "GO". Cette partie à pour but de présenter l'architecture de l'interface graphique qui a été codée à l'aide de la bibliothèque python *tkinter*.
Les codes relatifs à cette section se trouvent en ***Annexe 3***

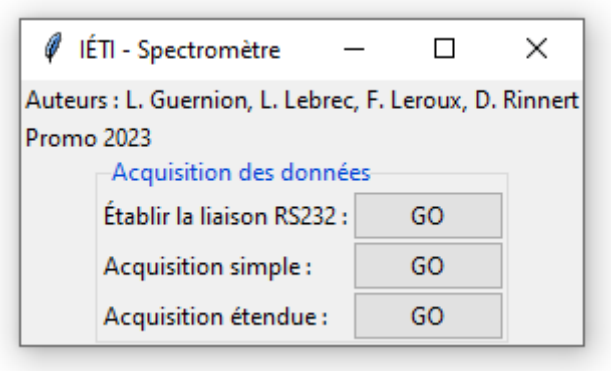


Figure 6 : Aperçu de l'interface graphique réalisée.

La **Figure 6** présente un aperçu de l'interface graphique. La création de cette dernière se fait par appel à la fonction *Tk()*, et nous lui attribuons un titre. Pour ce qui est de la gestion de l'espace au sein de l'interface graphique, nous procédons par la méthode *grid* qui décompose tout l'espace en une matrice de lignes et de colonnes. Ainsi pour afficher un *objet*, il suffit d'exécuter la ligne de code *objet.grid(row = IndiceDeLaLigne, column = IndiceDeLaColonne)*. Il existe d'autres arguments tels que *rowspan*, *columnspan*, ou *sticky* qui permettent de définir l'étalement sur plusieurs lignes et / ou colonnes, ou encore l'alignement de l'*objet* au sein de sa case.

Il existe plusieurs types d'*objet*, voici ceux que nous utilisons (nous les présentons uniquement de la manière dont nous nous en sommes servis, ils possèdent également d'autres applications) :

- *LabelFrame(InterfaceGénérale, text = LeTitreDeLaSousSection)* : Permet de créer une sous section dans l'interface générale.
- *Label(Interface, text = TexteÀAfficher)* : Il permet d'afficher un texte.
- *Button(Interface, command = FonctionAssociée, text = TexteÀAfficher)* : Permet d'exécuter une fonction qui lui est associée lorsque l'utilisateur clique dessus.

Remarque : Les différentes fonctions exécutables seront présentées par la suite.

Communication Nucléo - Machine, liaison série RS232

Par Damien Rinnert

La liaison RS232 nous permet de dialoguer avec la carte Nucléo via l'interface graphique en Python. Nous verrons dans cette partie deux fonctionnalités de la liaison RS232 :

- Établissement de la liaison série.
- Communication du tableau de 64 valeurs de la carte Nucléo vers la Machine.

Afin de réaliser ces deux fonctionnalités, la bibliothèque Python *pySerial* est nécessaire.

Établissement de la liaison série :

Les codes relatifs à cette section se trouvent en ***Annexe 4***

Avant toutes choses, il est nécessaire de lier la carte Nucléo et la Machine via le câble USB adapté. Afin de signaler au programme Python quel port utiliser pour communiquer, l'utilisateur doit cliquer sur le bouton "GO" associé à "Établir la liaison RS232" dans l'interface graphique. Ce dernier exécute la fonction *RS232_Function* qui permet d'afficher dans la console Python les différents ports disponibles via la fonction *serial_ports*. L'utilisateur doit alors entrer le numéro du port de communication associé à la carte Nucléo.

Une fois cette manipulation faite, la liaison série RS232 est établie et les informations sur la liaison sont stockées dans la variable globale Python *serialSelected*. Il n'y a donc plus besoin d'effectuer cette manipulation durant l'exécution du programme, à moins d'avoir changé de port USB.

Communication du tableau de 64 valeurs de la carte Nucléo vers la Machine :

L'algorithme présenté **Annexe 5** s'effectue en deux temps :

- La Machine ordonne à la Nucléo de réinitialiser son tableau de mesure, ainsi que la variable j (par l'envoie du caractère "m"), ce qui permet de réaliser une mesure sur 64 pixels (le caractère "o" est renvoyé à Python en guise de confirmation).
- Une fois que la mesure est terminée, on procède à l'envoie des données une à une vers le tableau mesure de Python : l'envoie du caractère "a" permet de transmettre l'intensité acquise par le pixel ind_mesure sous la forme de 2 bits d'indice, 1 bit d'espace, 8 bits d'intensité mesurée, puis 3 bits de mise en forme ("%2d %1.6lf \r\n") ; puis l'envoie du caractère "f" permet d'incrémenter ind_mesure de 1. Ce schéma est répété 64 fois afin de récupérer la totalité du tableau mesure dans le tableau mesure.

Remarque : Les erreurs de transmission via la liaison série sont prises en charge et déclenche un message d'erreur.

Les directives envoyées par Python sont gérées par deux fonctions :

Les codes relatifs à cette section se trouvent en ***Annexe 6***

- *sendingCmd* qui prend en argument un caractère codant la directive (ex : "m" ou "f"), un caractère attestant de la bonne transmission de la directive (ex : "o" ou "f"), puis un nombre de milliseconde maximal que l'on accorde pour la transmission des informations. Si la transmission a bien été effectuée dans le temps imparti, alors la fonction renvoie True, sinon elle renvoie False.
- *receivingData* qui prend en argument un caractère de commande (ex : "a") et puis un nombre de milliseconde maximal que l'on accorde pour la transmission des informations. Si la transmission a bien été effectuée dans le temps imparti, alors la fonction renvoie 14 bits sous la forme décrite précédemment, sinon elle renvoie False.

_____L'exécution de ces directives sur la carte Nucléo est gérée par la fonction ISR_get_data.

Remarque : Ces trois fonctions résultent de quelques modifications de codes fournies par Monsieur J. Villemejane que nous remercions.

Traitement des données, Python

Par Léo Guernion, Lancelot Lebrec, François Leroux, Damien Rinnert

Maintenant que nous sommes capable d'obtenir des mesures sur Python, il s'agit de savoir comment placer le goniomètre afin d'observer la partie du spectre qui intéresse l'utilisateur, combien de mesures doivent être effectuées, corriger les mesures en fonction de la sensibilité du capteur, et de mettre en forme les résultats afin d'afficher un spectromètre : l'intensité relative du spectre de la source en fonction de la longueur d'onde.

Dans cette partie nous présentons la procédure générale du traitement des données et de l'interaction avec l'utilisateur, ainsi que comment sont implémentés les deux modes de fonctionnement du spectromètre : *Acquisition simple* et *Acquisition étendue*.

La procédure générale est la suivante :

1) L'utilisateur indique à la machine des données techniques (pas du réseau, correspondances entre des longueurs d'onde de référence et les angles lus sur le goniomètre, plage angulaire couverte par la barrette optique).
2) Python calcule les données d'étalonnage qui permettent d'obtenir la correspondance entre l'angle du goniomètre et la longueur d'onde observée.
3) L'utilisateur définit la plage de longueur d'onde qu'il souhaite observer.
4) Python interagit avec l'utilisateur de façon à obtenir le spectre souhaité.
5) Les mesures sont corrigées en fonction de la sensibilité du capteur.

Détails de la procédure :
*Les codes relatifs à cette section se trouvent en **Annexe 7***

○ Récupération des données d'étalonnages

• Obtention de la plage angulaire couverte par le capteur

Afin d'obtenir la valeur $\Delta\theta$ de la plage angulaire couverte par la barrette optique, un protocole possible est le suivant :

- Visualiser une raie du spectre de la source à l'oscilloscope (branché en sortie de la carte Nucléo).
- Déplacer le goniomètre de sorte à visualiser le centre de cette raie sur le premier pixel de la barrette optique et noter la position angulaire.
- Déplacer à nouveau le goniomètre de sorte à visualiser le centre de la raie sur le dernier pixel de la barrette optique et noter cette nouvelle position.
- La différence en valeur absolue de ces deux positions est $\Delta\theta$.

• Procédure « angle-angle »

Le prototype de la fonction est le suivant :

Nom de la fonction :	procedure_angle_angle
Arguments d'entrée :	Aucun
Arguments de sortie :	$etalonnage = [\theta_1, \lambda_1, \Delta\lambda]$

Cette première méthode pour récupérer consiste à demander à l'utilisateur de se placer sur une première raie de de référence et d'indiquer la longueur d'onde de la raie ainsi que l'angle du

goniomètre correspondant, puis de faire de même pour une seconde raie de référence. Ainsi, on dispose d'un couple $\{(\theta_1, \lambda_1); (\theta_2, \lambda_2)\}$. À partir de là, en faisant l'approximation des petits angles, l'optique affirme qu'il y a une relation affine entre angle et longueur d'onde. On obtient donc la formule suivante qui donne le lien entre angle et longueur d'onde : $\lambda(\theta) = ((\lambda_2 - \lambda_1)/(\theta_2 - \theta_1)) \times (\theta - \theta_1) + \lambda_1$.

Pour des raisons pratiques, on défini les données d'étalonnages la le tableau suivant : $etalonnage = [\theta_1, \lambda_1, \Delta\lambda]$, où (θ_1, λ_1) est une correspondance angle / longueur d'onde. Et $\Delta\lambda$ est la différence entre la longueur d'onde sur le pixel gauche de la barrette optique et la longueur d'onde du pixel droit de la barrette optique.

• Procédure « pas-angle »

Le prototype de la fonction est le suivant :

Nom de la fonction :	procedure_pas_angle()
Arguments d'entrée :	Aucun
Arguments de sortie :	$etalonnage = [\theta_1, \lambda_1, \Delta\lambda]$

Cette deuxième méthode possède les mêmes arguments d'entrée et de sortie que la fonction précédente. Elle sert donc la même cause, à savoir récupérer les données d'étalonnage.

Pour cela, elle demande à l'utilisateur d'entrer une seule correspondance angle / longueur d'onde. Puis l'utilisateur doit entrer le pas du réseau (en traits par millimètre). Ces données sont suffisantes pour déduire les données d'étalonnage. En effet on possède un résultat d'optique, la formule des réseaux suivante : $\sin(\theta_m) = \sin(\theta_{incident}) - m \times \lambda/d$, où m est un entier désignant l'ordre de diffraction.

Ainsi, si l'on connaît dans quelle direction part une longueur d'onde, on peut en déduire l'angle de toutes les autres longueurs d'onde.

○ Deux modes de mesure

Maintenant que l'on a les données d'étalonnages, on souhaite graduer l'axe des abscisses en longueur d'onde.

• Cas d'une d'une seule prise de vue

Le prototype de la fonction est le suivant :

Nom de la fonction :	graduation	
Arguments d'entrée :	etalonnage	// données d'étalonnage
	grad	// angle lu par l'utilisateur sur le goniomètre
Arguments de sortie :	$[\lambda_g, \lambda_d]$	// tableau contenant les longueurs d'onde à
		// gauche et à droite de la barrette

Cette fonction peut être utilisée en tant que tel pour graduer l'axe des abscisses si une seule photo est prise.

• Cas de concaténation de plusieurs prises de vue

Le prototype de la fonction est le suivant :

Nom de la fonction :	spectre_utilisateur	
Arguments d'entrée :	etalonnage	// données d'étalonnage
Arguments de sortie :	$[k, \theta_{initial}]$	// k est le nombre de mesures a effectuer
		// $\theta_{initial}$ est l'angle de la première mesure

L'objectif de cette fonction est de définir la procédure pour récupérer spectre intéressant l'utilisateur lorsque la plage de longueur d'onde n'est pas accessible en une prise de vue.

Pour cela, on demande à l'utilisateur les longueurs d'onde maximales et minimales de la partie du spectre qu'il souhaite observer. On utilise ensuite les données d'étalonnage pour déterminer l'angle de la première mesure et le nombre de mesures à effectuer. La plage angulaire de la barrette optique étant une variable globale définie par l'utilisateur au préalable, il suffit donc de décaler le goniomètre de cet angle après chaque prise de vue.

○ Fonction finale

Le but est ici de compiler toutes les fonctions précédentes en une seule d'utilisation plus simple.
Le prototype de la fonction est le suivant :

Nom de la fonction :	info_mesure
Arguments d'entrée :	Aucun
Arguments de sortie :	$[k,\ \theta_{initial}, \lambda_g,\ \lambda_d]$

k est le nombre de mesures a effectuer ; $\theta_{initial}$ est l'angle de la première mesure ; λ_g et λ_d sont les longueurs d'onde à droite et à gauche du spectre complet.

Cette fonction fait appelle à *procedure_angle_angle* pour demander à l'utilisateur les données d'étalonnage. Puis elle utilise la fonction *spectre_utilisateur* pour déterminer la procédure à suivre par l'utilisateur. Enfin, elle fait appel à la fonction *graduation* pour déterminer les longueurs d'onde minimales et maximales du spectre final une fois les différentes prises de vue effectuées.

Le tableau de sortie $[k,\ \theta_{initial}, \lambda_g,\ \lambda_d]$ permet d'avoir rapidement accès à la procédure à suivre pour obtenir les spectre demandé ainsi que de pouvoir facilement graduer l'axe de abscisses du spectre final.

○ Correction d'intensité

Il se trouve que la barrette optique utilisée a une sensibilité qui dépend de la longueur d'onde. La documentation technique fournie la réponse ci contre du capteur en fonction de la longueur d'onde.

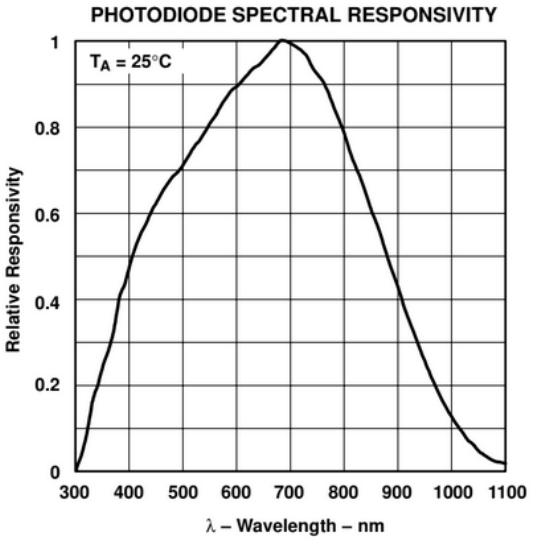


Figure 7 : Réponse spectral du capteur TSL201R
(extrait de la documentation technique de TAOS).

Nous avons donc approché cette courbe selon un modèle simple : deux droites dont nous avons déterminé les équations (nous ne nous intéressons qu'au domaine visible compris entre $400\ nm$ et $800\ nm$).

L'objectif est donc de corriger les données brutes mesurées par le capteur à l'aide de la modélisation de la réponse fréquentielle du capteur et de renvoyer le tableau des intensités corrigées. On a alors définit la routine qui suit (on note *n* la dimension du tableau *mesure* contenant les données brutes) :

```
mesure_corrigee = [None] * n
for k in range(n):
    if lambda_abs[k] > 675 :
        mesure_corrigee[k] = mesure[k]/(-0.0016 * lambda_abs[k] + 2.08)
    else:
        mesure_corrigee[k] = mesure[k] / (0.0018 * lambda_abs[k] - 0.22)
```

Réalisation d'une mesure :

Les codes relatifs à cette section se trouvent en ***Annexe 8***

Que ce soit pour une acquisition simple ou une acquisition étendue, il est nécessaire d'initialiser la mesure à une ou plusieurs reprises en fonction du mode d'acquisition. Cela consiste à ordonner à la carte Nucléo de réinitialiser son tableau de mesure ainsi que l'indice *j*, ce qui a pour conséquence d'initialiser une nouvelle série de mesure qui sera donc prête à être transmise à la machine. L'initialisation de la mesure se fait grâce à la fonction *Init* et la transmission grâce à la fonction *Acquérir_64*.

Présentation de l'acquisition simple :

Les codes relatifs à cette section se trouvent en ***Annexe 8***

Le bouton d'acquisition simple de l'interface graphique permet d'exécuter la fonction Python *Single_Function*. Elle permet d'obtenir un spectrogramme centré sur une raie intéressant l'utilisateur. Comme indiqué précédemment, son exécution débute par la demande d'informations nécessaires à l'étalonnage de l'appareil. Ensuite, le programme demande à l'utilisateur de centrer le capteur sur la raie qui l'intéresse, afin de procéder à la mesure. Cette mesure est ensuite corrigée selon la routine présentée précédemment. Ensuite, la connaissance de la plage angulaire couverte par le capteur ainsi que la relation $\lambda(\theta)$ nous permet de graduer l'axe des abscisses. Ce qui permet finalement d'afficher le spectrogramme brute ainsi que le spectrogramme corrigé.

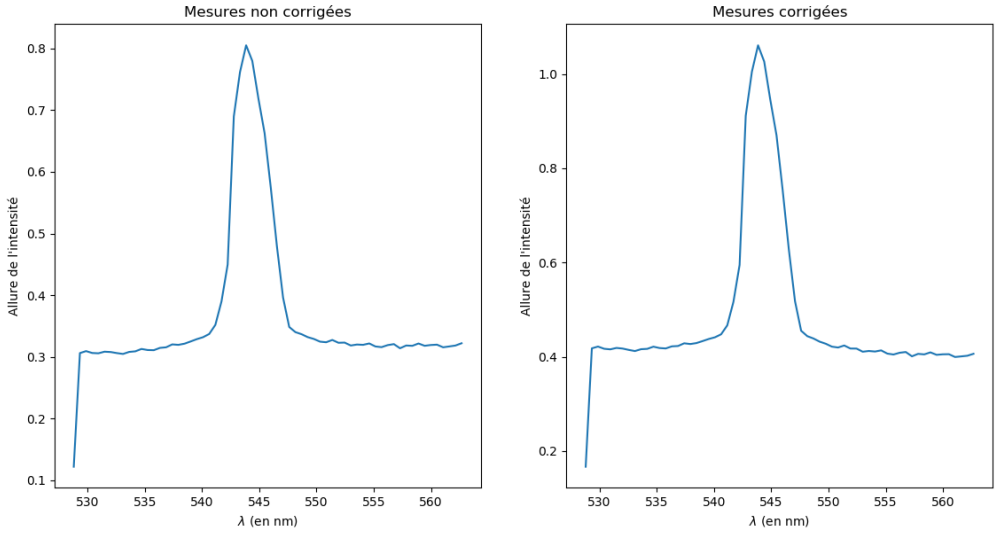


Figure 8 : Spectrogrammes obtenues pour une acquisition simple sur la raie jaune du mercure.

Présentation de l'acquisition étendue :

Les codes relatifs à cette section se trouvent en ***Annexe 8***

Le bouton d'acquisition simple de l'interface graphique permet d'exécuter la fonction Python *Acquérir_Function*. Cette fonction permet d'obtenir un spectrogramme sur une plage de longueur d'onde supérieure à celle couverte par le capteur en concaténant plusieurs mesures et en indiquant à l'utilisateur quelles sont les manipulations à effectuer. Lors de son exécution, les données d'étalonnages sont obtenues grâce à la fonction *info_mesure*, ce qui nous permet de définir un tableau *mesure_finale* de dimension 64 fois le nombre de mesure nécessaire *k* qui est complété par la répétition de mesures et de décalage $\Delta\theta$ sur le goniomètre (la procédure étant communiquée à l'utilisateur via la console Python). À l'issue de ces *k* mesures, nous pouvons procéder à la correction des mesures en fonction de la sensibilité du capteur et à l'affichage des résultats.

Remarque 1 : Nous noterons qu'il faut prêter attention à inverser les tableaux de mesures avant de les concaténer au reste des valeurs déjà acquises en raison, car en raison des conditions géométriques de notre montage, le premier pixel du capteur détecte une intensité pour une longueur d'onde plus grande que celle du dernier pixel du capteur.

Remarque 2 : Nous ne sommes pas parvenus à obtenir des résultats satisfaisants avec ce mode d'acquisition. Nous pouvons avancer des sources d'erreurs : le système optique n'étant pas très stable, il se peut que le capteur pivote lors des différentes mesures, changeant ainsi la valeur de $\Delta\theta$ indiqué en début d'acquisition et perturbant la continuité du signal aux différentes concaténations. Il semblerait également que le code prenne mal en charge la dernière mesure lors de la concaténation, pour des raisons que nous n'avons pas déterminé au bout de ces 8 séances. Cela représente néanmoins une piste d'amélioration du projet, ou bien en déterminant l'erreur actuelle (méthode préconisée), ou bien en adaptant le code de sorte à réaliser une mesure supplémentaire dans la boucle, mais qu'il ne faudrait pas prendre en compte pour l'affichage des résultats.

Rétroplanning / Difficultées rencontrées / Pistes d'améliorations

Rétroplanning

Séance 1 (25/01/2021) :

- Découverte du matériel, réflexion sur le cahier des charges et sur les moyens techniques à mettre en place

Séance 2 (01/02/2021) :

- Prise en main du capteur CMOS (alimentation par un GBF pour le déclenchement et l'horloge, et un générateur de tension continu, puis obtention d'un premier spectre à l'oscilloscope)
- Réalisation du cahier des charges et du premier schéma bloc

Séance 3 (10/03/2021) :

- Étude optique pour exploiter la dispersion du réseau et le goniomètre pour imager le spectre sur le capteur CMOS
- Écriture des premiers codes sous MBED pour piloter la carte Nucléo et générer les signaux de déclenchement et d'horloge
- Premier pas sur la connexion série RS232

Séance 4 (24/03/2021) :

- Obtention du premier spectre via la carte Nucléo
- Premiers codes python pour la connexion série RS232

Séance 5 (12/04/2021) :

- Test des codes python pour l'acquisition des données (communication série RS232)
- Élaboration de la procédure à suivre pour étalonner et manipuler le goniomètre et les programmes pythons associés
- Création de l'interface graphique (début)

Séance 6 (10/05/2021) :

- Audit de projet
- Finalisation la procédure d'étalonnage et les programmes pythons associés (inclut les mesures optiques nécessaires)
- Déboguer / Finaliser les programmes de communication série RS232
- Poursuite de l'interface graphique

Séance 7 (19/05/2021) :

- Derniers tests et ajustement ⇒ programmes finaux

- Affinage de l'interface graphique
- Préparation des livrables finaux

Séance 8 (26/05/2021) :

- Présentation finale du projet

Difficultés rencontrées

Partie optique :

Il nous a fallu trouver un bon moyen de diffracter le spectre de la source lumineuse. Le prisme posait des problèmes car la relation entre la longueur d'onde et l'angle d'observation n'est pas simple à manipuler. Nous nous sommes donc tournés vers les réseaux, mais il nous avons dû prêter attention au choix du pas du réseau afin que les différents ordres de diffraction ne se chevauchent pas et ne génèrent donc pas des raies parasites.

Partie acquisition / alimentation :

Afin de synchroniser la restitution des données par le capteur et leur lecture par le microcontrôleur, nous avions d'abord penser à effectuer des tests uniquement dans le code, type « if telle_sortie_de_la_carte = 1 ». En fait, ce n'est pas possible, il faut nécessairement boucler ces sorties de la carte destinées à alimenter le capteur sur des entrées analogiques de la carte pour effectuer les tests. Également, nous nous sommes rendu compte qu'il était impossible sous MBED de générer une interruption dans une interruption et avons dû restructurer nos codes en conséquence.

Remarquons également que la seule prise en main du capteur, c'est-à-dire réussir à le faire fonctionner avec le GBF et l'oscillo, et l'obtention de paramètres judicieux pour l'alimentation nous a pris une séance complète.

Partie interface graphique :

Très peu de problèmes rencontrées lors de la réalisation de l'interface graphique, si ce n'est que nous n'avons pas eu le temps de finir de développer un système de fenêtre PopUp afin de dialoguer avec l'utilisateur via l'interface graphique plutôt que par la console Python.

Partie communication série RS232 :

Au début nous ne disposions que de codes nous permettant de transmettre des caractères ASCII un à un. Ce qui nous posait des soucis car nous souhaitons transmettre des valeurs d'intensité décimale sur plusieurs bits. Nous nous sommes donc renseigné sur la bibliothèque pySerial afin de trouver comment communiquer plusieurs bits à la fois (avec l'aide de Monsieur J. Villemejane).

Il a également fallu pallier un second problème. Initialement lorsque la carte Nucléo transmettait les données, il n'y avait aucun moyen de les associer à tel ou tel pixel du capteur. Il nous a fallu restructurer le code afin de transmettre les valeurs du tableau de données une à une. Nous avons également introduit un formalisme pour la transmission des données afin de pouvoir identifier à quel pixel était associé la donnée transmise.

Partie traitement des données :

Nous avons rencontrées quelques difficultées liées à l'établissement de la valeur de $\Delta\theta$ qui est susceptible de varier au cours d'une mesure. Nous sommes malheureusement limités sur ce point par l'usinage du montage du goniomètre.

Il persiste encore les problèmes mentionnés auparavant à propos de l'acquisition étendue.

Améliorations potentielles

La partie optique du dispositif pourrait être améliorée, notamment en termes de rigidité du montage. Il est important que le capteur reste positionné dans le plan focale image de la lentille qui focalise la figure de diffraction pendant toute la durée des mesures, et qu'il ne tourne pas sur lui-même. De plus, il était difficile de lire les angles au goniomètre à cause des fils qui alimentaient le capteur. Cela permettrait probablement d'éviter les erreurs dans les résultats que nous obtenons lorsque l'on essaye d'obtenir le spectre complet de la source par déplacements successifs du capteur.

L'interface graphique pourrait également être améliorée de manière à pouvoir assurer la totalité de la communication avec l'utilisateur.

Enfin, au lieu de déplacer à la main le capteur lorsque l'on veut obtenir la totalité du spectre, il est envisageable d'utiliser un moteur pas à pas et d'inclure sa commande dans le code MBED et Python. Cela permettrait sans doute à notre système d'être plus autonome d'utilisation.

Annexes

Annexe 1 : Documentation de la barette optique

https://datasheet.octopart.com/TSL201R-LF-TAOS-datasheet-8327114.pdf

Annexe 2 : Programme de la carte Nucléo sous MBED

```
#include "mbed.h"

// inputs and outputs configuration
Serial      rs232(USBTX, USBRX);
PwmOut SI(D13);     // Signal de déclenchement
PwmOut CLK(D9);     // Signal clock, il détermine la fréquence de restitution de l'information
DigitalIn CLK_bool(D10);            // on branche CLK sur D10
AnalogIn ADC(A1);                 // entrée analogique de la carte nucléo = canal de sortie du capteur pour la restitution des 64 valeurs d'intensité à fclock
InterruptIn int_acquisition(A5);                // on branche SI sur DA5 : l'acquisition se déclenche sur un front montant du SI

// System functions
void ISR_get_data(void);
void acquisition();
// déclaration de la fonction qui fait la mesure : 64 valeurs correspondant à la moyenne sur N mesures de la valeur de l'intensité lumineuse reçue par chaque pixel

// Variables
#define N    100                    // nombre de mesures sur lesquelles on souhaite calculer la moyenne
double Tclk=1.0/20000.0;             // fclock=20kHz
int i=0;
// variable boucle for pour effectuer une mesure = remplir un tableau avec les 64 valeurs de l'intensité de chaque pixel à l'instant t PAR ADDITION SUCCESSIVE
int j=0;
// variable boucle for pour remplir N fois le tableau (permet calculer la moyenne)
int l;
// variable boucle for permettant d'afficher sur Tera-Term 64 valeurs correspondant à la valeur de l'intensité de chaque pixel moyennée sur N mesures
int test=1;
// permet de s'assurer de ne pas passer plusieurs fois dans la boucle effectuant la mesure de l'intensité reçue par UN pixel au cour d'un tps haut de la clock


char data_received = 0;
double mesure[64];
int ind_mesure; // Il sert à parcourir tableau de mesure


// Main function
int main() {
    rs232.baud(115200);              // Définition de la liaison série
    rs232.attach(&ISR_get_data);     // Attache de la procédure de communication à la liaison série
    CLK.period(Tclk);                // programmation des tensions d'alimentation
    SI.period(100*Tclk);
    CLK.write(0.5);
    SI.write(0.01);
    for (ind_mesure = 0 ; ind_mesure < 64 ; ind_mesure ++)
        mesure[ind_mesure] = 0 ;
    j = N;
    int_acquisition.rise(&acquisition);    // un front montant du SI => une acquisition de 64 mesures
    while(1) {}
}
```

```
void acquisition()
{
    while (i<64 && j<N)                 // une mesure parmi N
    {
        if (CLK_bool==1 && test==1)
        {
            test = 0;
            mesure[i] += ADC.read()*3.3;
            i++ ;
        }
        if (CLK_bool==0){test=1;}

    }
    if (i==64){i=0;}                     // on passe le dernier front montant
    if (j==N)                            // on vient de finir de faire les N mesures
    {
        for(l=0 ; l<64 ; l++)
            mesure[l] = mesure[l]/N;
        ind_mesure = 0;
    }
    j++ ;
}
```

Annexe 3 : Gestion de l'interface graphique

```
# Création de l'interface
app = Tk()
app.title("IÉTI - Spectromètre")
auteurs = Label(app, text = "Auteurs : L. Guernion, L. Lebrec, F. Leroux, D. Rinnert")
auteurs.grid(row = 0, column = 0, columnspan = 1)
date = Label(app, text = "Promo 2023")
date.grid(row = 1, column = 0, columnspan = 1, sticky = "w")

# Création du bloc d'acquisition des données
acq = LabelFrame(app, text="Acquisition des données")
acq.grid(row = 3, column = 0, rowspan = 3, columnspan = 2)

RS232_Label = Label(acq, text="Établir la liaison RS232 :")
RS232_Label.grid(row = 0, column = 0, sticky = "w")
RS232_Button = Button(acq, command = RS232_Function, text = "GO")
RS232_Button.grid(row = 0, column = 1, sticky = "e")

Single_Label = Label(acq, text = "Acquisition simple :")
Single_Label.grid(row = 1, column = 0, sticky = "w")
Single_Button = Button(acq, command = Single_Function, text = "GO")
Single_Button.grid(row = 1, column = 1, sticky = "e")

Acquérir_Label = Label(acq, text = "Acquisition étendue :")
Acquérir_Label.grid(row = 2, column = 0, sticky = "w")
Acquérir_Button = Button(acq, command = Acquérir_Function, text = "GO")
Acquérir_Button.grid(row = 2, column = 1, sticky = "e")

app.mainloop()
```

Annexe 4 : Établissement de la liaison série

```
def serial_ports():
    '''
        USE : serial_ports()
        This function aims at printing all available ports.
    '''
    nb_port = 0
    ports = list(serial.tools.list_ports.comports())
    list_port = []
    for port in ports:
        print(port)
    return

def RS232_Function(*args) :
    '''
        USE : RS232_Function()
        This function aims at initialising the serial connexion between the computer and the Nucelo.
    '''
    global serialSelected
    serial_ports()
    serialPortSelected = input('Enter the Nucleo port: ')
    serialSelected = serial.Serial('COM' + serialPortSelected, 115200, timeout = 1)
    return
```

Annexe 5 : Présentation de l'algorithme de communication du tableau de mesure.

Projet spectromètre

Liaison série RS232
Communication Python - MBED

Variables

N : nombre de mesures sur lesquelles le signal est moyenné

mesure : tableau de 64 *doubles* contenant les données de l'intensité pour chaque pixel de la barrette optique (MBED)

ind_mesure : indice permettant de parcourir le tableau mesure

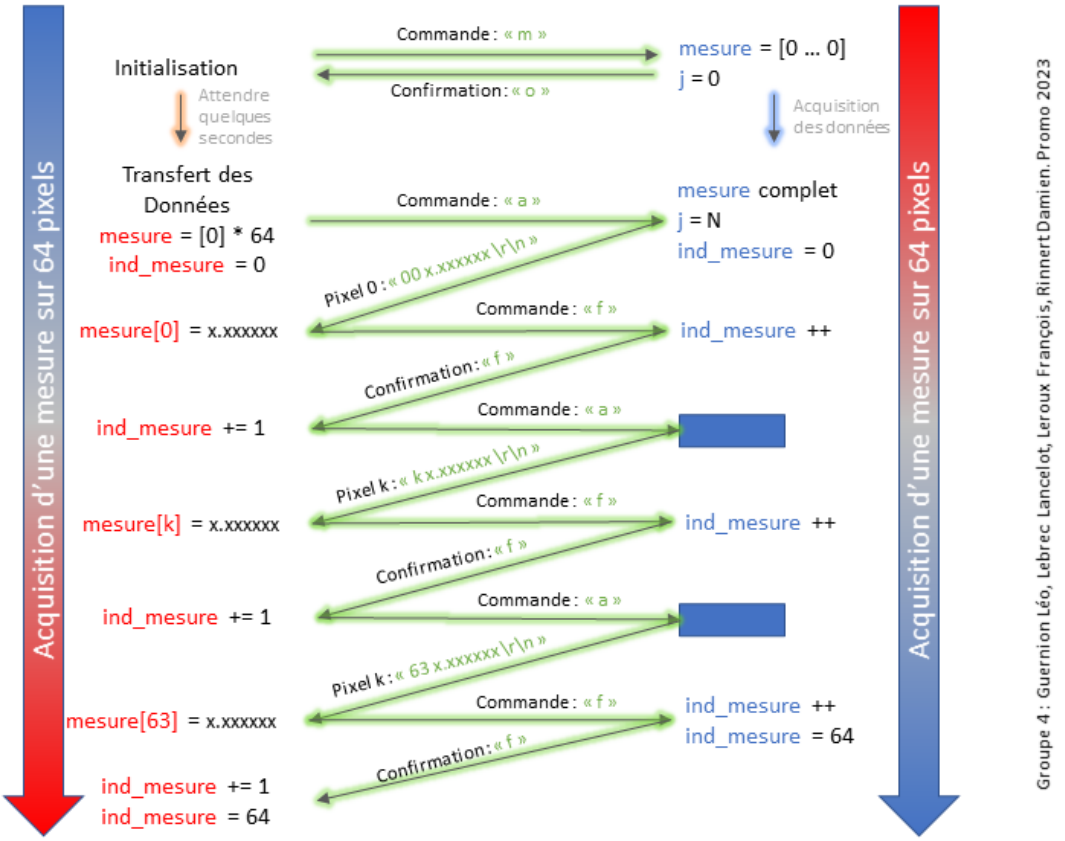
j : variant de boucle représentant le nombre de mesures effectuées (j = 0 ➔ enclenchement de la première mesure ; j = N ➔ fin des mesures)

mesure : tableau de 64 *doubles* contenant les données de l'intensité pour chaque pixel de la barrette optique (Python)

ind_mesure : indice permettant de parcourir le tableau mesure

Table de communication (Python - MBED)		
Python	MBED	Action
"m"	"o"	Initialisation de la mesure : Réinitialisation du tableau de mesure MBED puis du variant de boucle j
"a"	"%2d %1.6lf \r\n"	Réception d'une donnée si ind_mesure < 64
"f"	"f"	Incrémente ind_mesure de 1 si ind_mesure < 64

Algorithme



Annexe 6 : Communication via la liaison série RS232

```
def sendingCmd(cmd, test, timeout_ms):
    '''
        USE : sendingCmd(cmd, test, timeout_ms).
        Entry :
            - cmd : Name of the command we want to be execute by the Nucleo.
            - test : Name of the answer we are waiting from the Nucleo.
            - timeout_ms : Number of ms we allowed to get an answer.
        Return :
            - True if the command has been send successfully.
            - False if the command has not been send successfully.
    '''
    global serialSelected
    if(serialSelected.isOpen() == False):
        serialSelected.open()
    serialSelected.write(bytes(cmd,'utf-8'))
    dataRecOk = 0
    timeoutRec = 0
    while (dataRecOk == 0) or (timeoutRec < timeout_ms):
        timeoutRec += 1
        if(timeoutRec > timeout_ms):
            return False
        if serialSelected.inWaiting() > 0:
            dataReceived = serialSelected.read(1)
            if(dataReceived == bytes(test,'utf-8')):
                dataRecOk = 1
                return True
            else:
                return False
        time.sleep(0.001)
    return False

def receivingData(cmd, timeout_ms):
    '''
        USE : receivingData(cmd, timeout_ms).
        Entry :
            - cmd : Name of the command which ask informations to the Nucleo.
            - timeout_ms : Number of ms we allowed to get an answer.
        Return :
            - Datas which have been received if it worked.
            - False if the operation went wrong.
    '''
    global serialSelected
    if(serialSelected.isOpen() == False):
        serialSelected.open()
    serialSelected.write(bytes(cmd,'utf-8'))
    timeoutRec = 0
    while (timeoutRec < timeout_ms):
        timeoutRec += 1
        if(timeoutRec > timeout_ms):
            print ("We didn't achieved to receive datas.\n")
            return False
        if serialSelected.inWaiting() > 0:
            dataReceived = serialSelected.readline()
            return dataReceived
        time.sleep(0.001)
    return False
```

```
void ISR_get_data(){
    data_received = rs232.getc();
    switch(data_received){
        case 'm':
            rs232.putc('o');
            for (ind_mesure = 0 ; ind_mesure < 64 ; ind_mesure ++)
                mesure[ind_mesure] = 0 ;
            j = 0;
            break;
        case 'a':
            if(ind_mesure < 64)
                rs232.printf("%2d %1.6lf \r\n", ind_mesure, mesure[ind_mesure]);
            break;
        case 'f':
            if(ind_mesure < 64)  ind_mesure+=1;
            rs232.putc('f');
            break;
        default:
            rs232.putc('k');
    }
}
```

Annexe 7 : Procédure d'étalonnage

```
def procedure_angle_angle():
    global Delta_theta
    t1 = float(input("Entrer l'angle (en °) de la premiere raie de référence: "))
    lbd1 = float(input("Entrer la longueur d'onde la premiere raie de référence (en nm) : "))
    t2 = float(input("Entrer l'angle (en °) de la seconde raie de référence : "))
    lbd2 = float(input("entrer la longueur d'onde la seconde raie de référence (nm) : "))
    dl = abs(Delta_theta*(lbd1-lbd2)/(t1-t2))
    etalonage = [t1,lbd1,dl]
    return (etalonage) # données d'étalonage décrivant comment placer le réseau sur le goniomètre

def graduation(etal,grad):
# Fonction intermédiaire qui renvoie la longueur donde correspondante à gauche et à droite de la barrette optique.
    global Delta_theta
    lambdag=(grad-etal[0])*etal[2]/Delta_theta+etal[1]
    return([lambdag,lambdag+etal[2]])

def spectre_utilisateur(etal):
    # fonction intermédiaire qui renvoie le nombre de mesures a effectuer et langle de la première mesure
    global Delta_theta
    lambdag=float(input("Entrer la plus petite longueur d'onde du spectre à observer (nm) : " ))
    lambdad=float(input("Entrer la plus grande longueur d'onde du spectre à observer (nm) : " ))
    k = int((lambdad-lambdag)/etal[2]) + 1
    teta_initial=Delta_theta+etal[0]+(Delta_theta/etal[2])*(lambdag-etal[1])
    return([k,teta_initial])

def info_mesure():
    global Delta_theta
    Delta_theta = float(input("Entrer la valeur de la plage angulaire correspondant à la barette optique, Delta_theta (en °) : "))
    etal=procedure_angle_angle()
    L=spectre_utilisateur (etal)
    lg=graduation(etal,L[1]-Delta_theta)[0]
    ld=graduation(etal,L[1]+(L[0]-1)*Delta_theta-Delta_theta)[1]
    return(L+[lg,ld])


def procedure_pas_angle():
    #procédure qui renvoie les données détalonage du goniometre par une seconde méthode
    teta=float(input("entrer l'angle de la raie"))+1.5
    lbd=float(input("entrer la longueur d'onde la raie (nm)"))
    pas=float(input("rentrer le pas du reseau (traits/mm)"))
    d=(1/pas)*(10**-3)
    diff=(360-teta)*(3.141592/180)
    tetacapt=3*(3.141592/180)
    x=(lbd*(10**-9)/d)*0.5*(1/sin(diff/2))
    somme=2*acos(abs(x))
    lbd_prime=(sin((somme+diff)/2+tetacapt)-sin((somme-diff)/2))*d
    dl=abs(lbd-abs(lbd_prime*(10**9)))
    etalonage=[teta-1.5,lbd,dl]
    return etalonage
```

Annexe 8 : Code relatifs aux mesures et acquisitions

```
def Init() :
    '''
        USE : Init()
        This function aims at launching a new acquisition according to the defined procedure.
    '''
    if sendingCmd('m', 'o', 20) == False:
        messagebox.showerror("Error", "Impossible d'initialiser la mesure. \n \rVérifier la connexion RS232.")
    else:
        print("Acquisition launched.\n")
    return
```

```
def Acquérir_64() :
    global serialSelected
    #Décalaration du tableau contenant les informations à traiter
    mesure_64 = [None]*64
    compteur = 0
    # Configuration du port de liaison
    if(serialSelected.isOpen() == False):
        serialSelected.open()
    # Sending char 'a' at the beginning and wait for string
    while (compteur < 64) :
        dataR = receivingData('a', 20)
        indice = int(dataR[0:2])
        valeur = float(dataR[4:12])
        if(indice == compteur):
            mesure_64[compteur] = valeur
            if(sendingCmd('f', 'f', 10)):
                compteur += 1
        else:
            messagebox.showerror("Erreur", "Erreur indice.\n\r")
            return
    return mesure_64
```

```
def Single_Function(*args) :
    '''
        Cette fonction a pour but d'établir le spectromètre d'une source autour d'une raie de longueur d'onde connue
    '''
    # Acquisition des données d'étalonnage
    global Delta_theta
    Delta_theta = float(input("Entrer la valeur de la plage angulaire correspondant à la barette optique, Delta_theta (en °) : "))
    print("Nous procédons à l'étalonnage de l'appareil avec deux raies de référence.")
    lambda_ref_1 = float(input("Entrer la longueur d'onde de la première raie de référence (en nm) : "))
    theta_ref_1 = float(input("Entrer la position angulaire de la première raie de référence (en °) : "))
    lambda_ref_2 = float(input("Entrer la longueur d'onde de la seconde raie de référence (en nm) : "))
    theta_ref_2 = float(input("Entrer la position angulaire de la seconde raie de référence (en °) : "))
    # Acquisition de la longueur d'onde étudiée
    lambda_e = float(input("Entrer la longueur d'onde que vous souhaitez étudier (en nm) : "))
    # Création de l'axe des abscisses
    alpha = abs((lambda_ref_2 - lambda_ref_1)/(theta_ref_2 - theta_ref_1)) # Coefficient directeur de la droite lambda = f(theta) (cf. théorie optique)
    lambda_abs = [(-32 + i) * Delta_theta / 64 * alpha + lambda_e for i in range(64)]
    # Réalisation de la mesure
    print("Veiller à ajuster la raie sur le centre de la barette optique")
    input('''Veuillez presser "Entrer" une fois que cela est fait.''')
    Init()
    time.sleep(3)
    mesure = Acquérir_64()
    # Correction de la mesure
    mesure_corrigee = [None] * 64
    for k in range(64):
        if lambda_abs[k] > 675 :
            mesure_corrigee[k] = mesure[k]/(-0.0016 * lambda_abs[k] + 2.08)
        else:
            mesure_corrigee[k] = mesure[k] / (0.0018 * lambda_abs[k] - 0.22)
    # Affichage des courbes
    plt.figure()
    plt.subplot(1,2,1)
    plt.plot(lambda_abs, mesure)
    plt.title("Mesures non corrigées")
    plt.xlabel("$\lambda$ (en nm)")
    plt.ylabel("Allure de l'intensité")
    plt.subplot(1,2,2)
    plt.plot(lambda_abs, mesure_corrigee)
    plt.title("Mesures corrigées")
    plt.xlabel("$\lambda$ (en nm)")
    plt.ylabel("Allure de l'intensité")
    plt.show()
    return
```

```
def Acquérir_Function(*args) :
    global Delta_theta
    # Obtention des variables utiles à la mise en place de la procédure avec l'utilisateur
    # On note k le nombre de mesures à effectuer pour obtenir le spectre complet
    [k, theta_init, lambda_init, lambda_final] = info_mesure()
    # Déclaration du tableau de résultat final et de l'axe des abscisses en longueur d'onde
    mesure_finale = [None] * 64 * k
    lambda_abs = np.linspace(lambda_init, lambda_final, 64*k)

    # Acquisition
    for i in range(k):
        Init()
        print("Placer vous à $\\theta$ = " + str(theta_init + i * Delta_theta) + "° puis cliquer sur OK.\n")
        input('''Presser "Entrer" une fois que c'est fait.\n''')
        print("Mesure en cours\n")
        mesure_finale[i*64:(i+1)*64] = list(reversed(Acquérir_64()))
        # On inverse l'acquisition afin d'assurer la bonne liaison de deux mesures consécutives (en raison de l'orientation du capteur)
    print("Fin de l'acquisition\n")

    # Correction des mesures en fonction de la sensibilité de la barette optique
    mesure_finale_corrigee = [None] * 64 * k
    for k in range(64 * k):
        if lambda_abs[k] > 675 :
            mesure_finale_corrigee[k] = mesure_finale[k] / (-0.0016 * lambda_abs[k] + 2.08)
        else:
            mesure_finale_corrigee[k] = mesure_finale[k] / (0.0018 * lambda_abs[k] - 0.22)

    # Affichage du spectromètre
    plt.figure()
    plt.subplot(1,2,1)
    plt.plot(lambda_abs, mesure_finale)
    plt.title("Mesures non corrigées")
    plt.xlabel("$\lambda$ (en nm)")
    plt.ylabel("Allure de l'intensité")
    plt.subplot(1,2,2)
    plt.plot(lambda_abs, mesure_finale_corrigee)
    plt.title("Mesures corrigées")
    plt.xlabel("$\lambda$ (en nm)")
    plt.ylabel("Allure de l'intensité")
    plt.show()
    return
```