LEBOUL Florian
Année universitaire 2016-2017

DEPARTEMENT INFORMATIQUE
Institut Universitaire de Technologie
8, rue Montaigne
PB 561
56000 Vannes cedex

Rapport de STAGE
de fin d'études

« Interfaçage d'un spectrofluorimètre »

Stage effectué au sein du département de recherche Sciences de la Matière et de la Vie à l'Université Bretagne Sud

Du 10 avril au 16 juin 2017

Maître de stage : M Olivier SIRE
Université Bretagne Sud : rue André Lwoff, 56000 Vannes
Enseignant tuteur, IUT de Vannes : M Régis Fleurquin

Remerciements

Je tiens à remercier M. Olivier Sire, mon maître de stage, pour avoir eu la confiance nécessaire en moi pour m'accepter en tant que stagiaire, mais également pour m'avoir intégré à l'équipe d'enseignants chercheurs présents dans le bâtiment du stage. Je le remercie également de la grande liberté de choix techniques qu'il m'a laissé pour ce stage, ce qui m'a permis d'apprendre à utiliser de nouveaux équipements, ainsi que comprendre le fonctionnement d'une machine sur laquelle, sans ce stage, je n'aurais jamais eu l'occasion de travailler. Je tiens également à remercier l'équipe d'enseignants chercheurs pour sa sympathie, et l'aide qu'elle m'a apporté.

En second lieu, je tiens à remercier M. Fleurquin tant pour l'intérêt porté au stage que pour les conseils qu'il m'a donnés afin de réussir ce dernier.

Je tiens également à remercier les enseignants de l'IUT pour m'avoir préparé et enseigné les fondamentaux du relationnel en entreprise, ce qui m'a permis de m'intégrer dans une équipe déjà existante. Je tiens aussi à les remercier pour la préparation que nous avons eue aux entretiens téléphoniques et physiques. Enfin, je les remercie de nous avoir transmis les connaissances et la logique nécessaire pour pouvoir réaliser des applications dans divers langages de programmation

Table des matières

1. Introduction

a. Présentation de l'entreprise

i. L'Université Bretagne Sud

L'Université Bretagne-Sud (UBS), créée en 2000 et actuellement présidée par Jean Peeters, comporte six composantes, chacunes réparties sur les campus de Vannes et de Lorient. C'est une université pluridisciplinaire qui forme en moyenne 8500 étudiants par an. Elle est également membre de plusieurs pôles de recherches, comme par exemple l'université européenne de Bretagne dont elle est la cofondatrice, ou encore le réseau des universités de l'Ouest Atlantique.

Le stage s'est déroulé sur le campus de Tohannic, à Vannes. C'est en 2000 que les premiers bâtiments furent ouverts, suivis de plusieurs autres dans les années qui suivirent. Ils regroupent petit à petit plusieurs enseignements, et accueillent l'UFR Droit, Sciences Économiques et de Gestion et l'IFSI. C'est d'ailleurs sur ce campus qu'un rapprochement entre les formations infirmières et universitaires a eu lieu pour la première fois en France. Aujourd'hui le campus dispose de plusieurs unités de recherches, toutes regroupées dans le Centre d'enseignement et de recherche Yves Coppens, et dans le laboratoire LabSTICC dans l'ancien IUP. Ce campus présente également une bibliothèque universitaire depuis 2004 et un restaurant universitaire depuis 2011. Quatre grands types d'enseignement y sont centralisés : les Facultés de Droit, Sciences Économiques et Gestion (DSEG), l'École Nationale Supérieure d'Ingénieurs de Bretagne-Sud (ENSIBS), l'Institut de Formation aux Soins Infirmiers (IFSI), ainsi que la Faculté des Sciences et Sciences de l'Ingénieur (SSI). C'est précisément dans cette dernière que le stage a eu lieu. La Faculté SSI est dirigée par Frédéric Bedel depuis novembre 2012. Elle est à la charge des formations en Physique, Chimie, Sciences de l'Ingénieur, Biologie, Environnement, Mathématiques, Sciences et Techniques Médico-Sociales, ainsi qu'Informatique. Ces formations sont structurées en trois départements. Le département Sciences et Techniques (SET), le département Sciences de la Matière et de la Vie (SMV), et le département Mathématiques Informatique et Statistique (MIS). La Faculté comporte également deux unités de recherche, le pôle MathSTIC, et le pôle TMV.

ii. Maître de stage

Mon maître de stage est le professeur Olivier Sire, professeur de biophysique qui anime l'équipe de biologistes et physico-chimistes de Vannes avec une antenne à Pontivy. Il est rattaché à l'IRDL qui est en cours de reconnaissance comme Unité Mixte de Recherche CNRS. Concernant l'enseignement, il contribue notamment aux enseignements de biophysique et donc de spectroscopie. C'est à ce titre qu'il a décidé, avec le concours de l'UFR de faire interfacer le spectrofluorimètre SLM qui lui a été légué par son ancien directeur de recherche, Bernard Alpert, aujourd'hui retraité.

b. Présentation du stage

i. Le contexte du stage

L'Université Bretagne Sud base la formation de ses étudiants en formation scientifique sur une grande partie de pratique, dans des salles de TP. Ce choix permet de former des étudiants capables de pratiquer, même avant l'obtention du diplôme. En contrepartie, les machines devant équiper ces salles représentent un budget très élevé. Il n'est donc pas possible d'en changer tous les ans.

Il y a plusieurs années, un enseignant chercheur partant à la retraite a voulu faire don à l'UBS d'un spectrofluorimètre. Ce type d'appareil permet d'étudier la fluorescence de solutions liquides provenant de molécules biologiques ou de synthèse. C'est-à-dire que l'on difracte une source lumineuse dite blanche ou polychromatique pour sélectionner une seule longueur d'onde la composant. Cette opération optique est effectuée par un monochromateur dont l'élément optique actif est un réseau de diffraction. On envoi cette lumière d'excitation sur un échantillon à tester. Ensuite, on réutilise un autre monochromateur (dit d'émission) pour effectuer l'analyse spectrale (résolue en longueur d'onde) de la lumière émise par le ou les fluorophores de l'échantillon, que l'on mesure par le courant collecté par une photodiode. Cet appareil, datant de 1978 dispose d'une optique irréprochable, d'une rare qualité, et inégalée jusqu'à aujourd'hui. En contrepartie, l'optique ayant bien vieilli, la partie électronique, ainsi que le logiciel prévu pour le brancher à un PC ont été totalement dépassés par les améliorations technologiques.

À la vue du prix d'un tel appareil, l'UBS a accepté ce don, et a donc récupéré ce spectrofluorimètre, aujourd'hui inutilisable en l'état, sans interface. C'est pourquoi la recherche d'une personne pouvant recréer de quoi faire fonctionner cette machine est devenue une obligation pour pouvoir l'utiliser.

ii. Tâche attribuée – Cahier des Charges

La tâche qui m'a été attribuée était donc simple en soi, créer une interface pour le spectrofluorimètre. Il m'a été imposé de créer un graphique, avec des vumètres pour observer les valeurs des capteurs, et d'instaurer un système de sauvegarde. J'ai été totalement libre sur le choix des technologies et langages de programmation utilisés.

iii. Fonctionnement du spectrofluorimètre

Un spectrofluorimètre est un appareil de mesure de fluorescence, c'est-à-dire une émission de lumière par des molécules qui ont été préalablement excitées par un rayonnement lumineux. Le principe est d'irradier un échantillon par une lumière monochromatique et de mesurer le spectre de cette molécule en fonction de la longueur d'onde. Cette fluorescence apporte de nombreuses informations sur la molécule, son environnement ou encore les partenaires moléculaires avec lesquels elle est susceptible d'interagir en formant des complexes. Plus en détail, si l'on considère le fonctionnement de cette machine comme une chaîne, au début il y a un arc électrique à l'argon qui produit une lumière blanche (large spectre). Cette lumière est ensuite soumise à un monochromateur d'excitation relié à un moteur pas à pas configurable via l'interface de commande. Ce Monochromateur est un réseau de diffraction de lumière, qui permet de sélectionner une longueur d'onde du spectre difracté (en lumière visible dans notre cas). Ce rayonnement passe par une fente optique (fente de sortie du monochromateur) et vient irradier l'échantillon disposé dans une cuve optique de 1 cm de côté. Chaque position du moteur correspond à une longueur d'onde. En recevant la lumière, l'échantillon fluoresce et devient donc une source lumineuse. Ensuite, la lumière émise par l'échantillon passe dans un second couple monochromateur/fente, afin d'en mesurer séquentiellement l'intensité longueur d'onde par longueur d'onde. Cette dernière est mesurée par une photodiode, simple capteur de luminosité (cf image 1). Le résultat est un spectre d'émission de fluorescence.

Image 1 : Schéma de fonctionnement du spectrofluorimètre

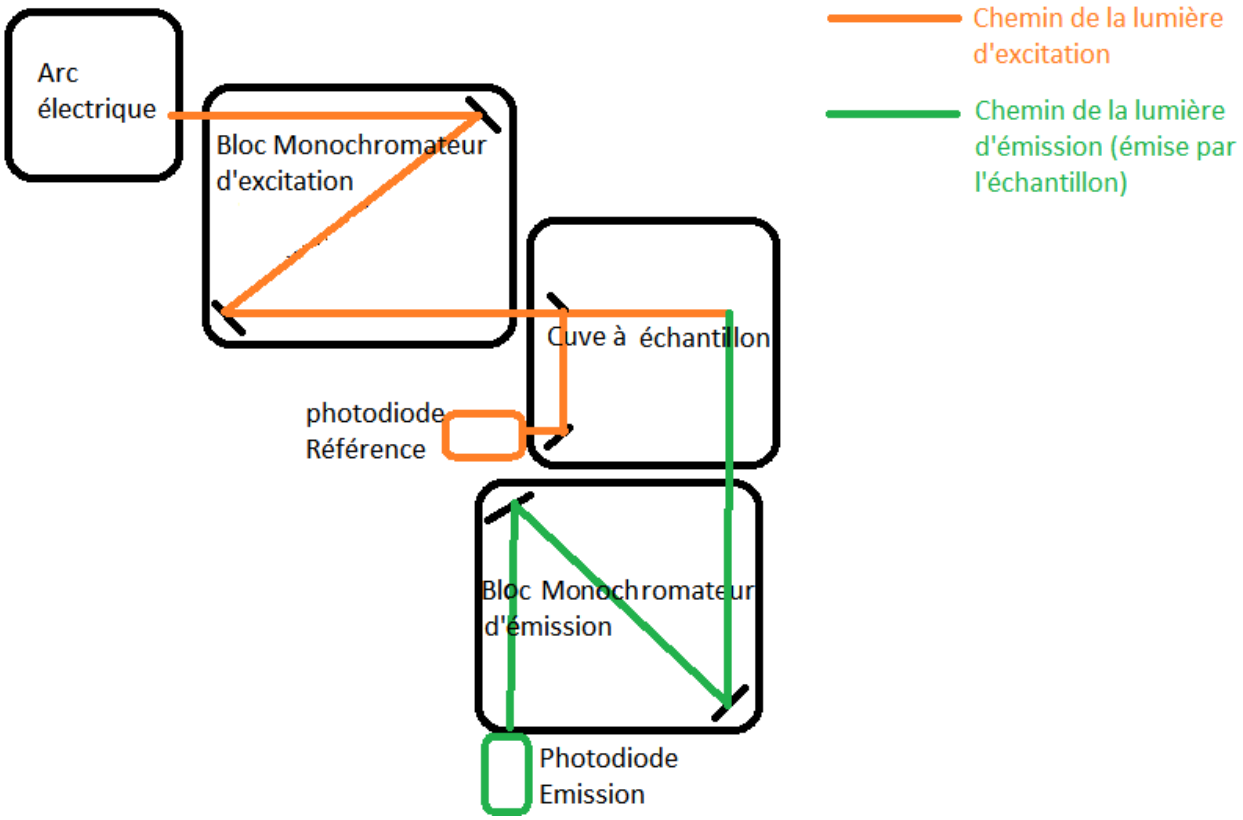
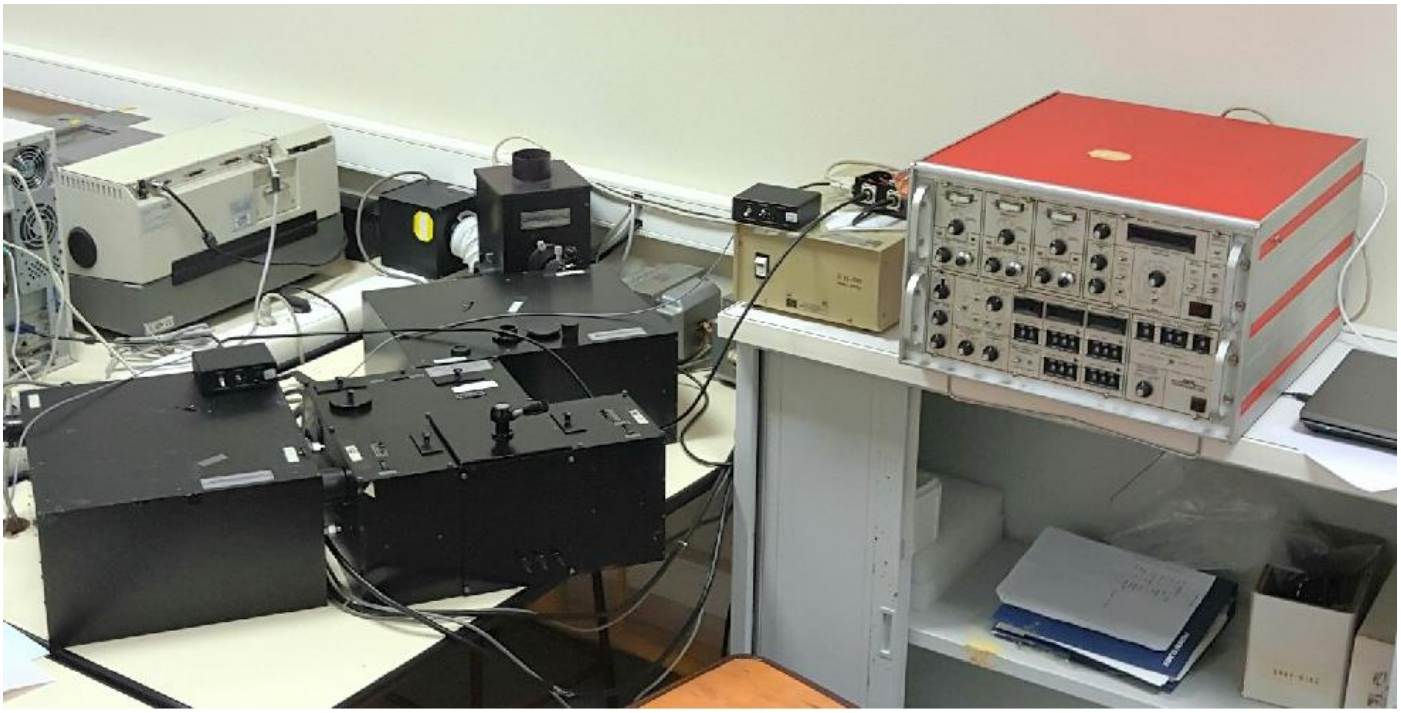


Image 2 : Spectrofluorimètre sur le terrain

Ci-dessous, l'installation sur laquelle j'ai effectué mon stage. Le spectrofluorimètre occupe toute la partie gauche (en noir). L'arc électrique est amorcé dans le boitier avec une cheminée, à l'arrière-plan. La partie située juste devant lui contient le monochromateur d'excitation (qui difracte la lumière de l'arc). Le boitier rattaché à son front contient l'emplacement pour insérer l'échantillon, et la dernière partie, à gauche de ce dernier, contient le monochromateur d'émission (qui diffracte la lumière de l'échantillon).



2. Partie I : électronique

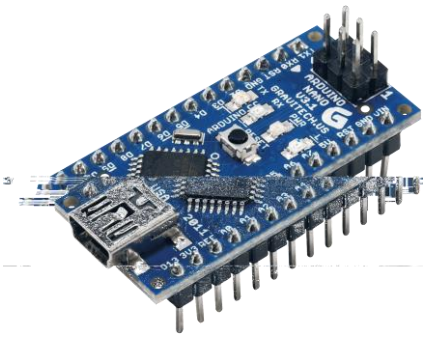
La première partie de ce projet, la plus importante, est l'électronique. Les anciennes machines comme le spectrofluorimètre utilisé lors du stage peuvent être sophistiquées. Mais elles sont aujourd'hui dépassées par les évolutions en matière d'informatique. L'électronique permet de palier ce décalage.

a. Au cœur de l'électronique : le microcontrôleur

L'intérêt, dans ce projet, d'intégrer un microcontrôleur réside dans la possibilité de restructurer les informations. C'est-à-dire qu'il n'est pas obligatoire d'utiliser les informations envoyées par la machine, ni le système de communication original. On peut choisir ce que l'on veut récupérer, et comment l'envoyer.

Le choix du microcontrôleur a rapidement tendu vers un Arduino, pour son rapport qualité/prix élevé, sa grande documentation et sa communauté d'aide surdéveloppée. Dans la mesure où le support technique pour toute la partie spectrofluorimètre était inexistant, j'ai préféré opter pour une technologie avec support développé. L'avantage de ces microcontrôleurs, hormis leur prix, est la grande diversité de modèles. Après comparaison, j'ai opté pour une carte Arduino nano, opérant à 5V via une prise USB, disposant d'un encombrement réduit (cf image 3).

Image 3 : Microcontrôleur Arduino Nano



b. Communication Spectrofluorimètre – Arduino :

Pour la communication entre les deux appareils, deux solutions se sont concurrencées. Le langage utilisé est semblable sur les deux systèmes. Tous deux communiquent par trames binaires (suite de 0 et de 1), seuls les niveaux logiques diffèrent. En effet, le spectrofluorimètre utilise la norme RS232, et l'Arduino, la norme TTL (cf image 4). Ces signaux RS232 sont non seulement incompréhensibles pour un microcontrôleur qui communique en TTL, mais également dangereux pour l'intégrité de ce dernier, du fait de possibles surtensions.

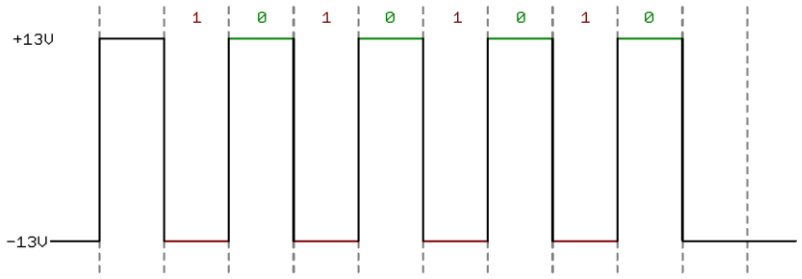
La première idée était d'utiliser un transistor en guise de relais (cf image 5). Cette solution n'était pas parfaitement adaptée, du fait de l'imprécision du transistor. Lors du passage d'un état à l'autre, on obtenait des pics de tension dangereux pour la carte. La seconde idée a donc été de passer par un circuit intégré externe, le MAX232 (cf image 6), permettant de convertir le RS232 en TTL. Ce second choix est plus exploitable, étant initialement dédié à cet usage.

N'ayant pas pu pratiquer dès le début du stage, du fait que nous ne savions pas démarrer la machine, je me suis basé sur de la théorie, ne sachant pas quelles données seraient envoyées par le spectrofluorimètre. Je disposais néanmoins du code source de l'application originale, codée en BASIC. Il m'a fallu quelques temps pour comprendre la syntaxe. En revanche, cette étude ne m'a pas réellement aidé dans la mesure où l'utilisateur entrait les réglages, et une seule courbe était tracée au fur et à mesure. Plus tard, lorsque nous avons réussi à démarrer la machine, j'ai compris le sens de ce dernier. La valeur affichée sur le graphique dépend d'un bouton rotatif sur l'appareil de commande. Aucune demande d'information n'est réalisée par le PC, il affiche simplement les valeurs reçues. Nous nous sommes rapidement aperçus que les informations envoyées manquaient d'utilité. La machine dispose de deux voies de capteurs utilisées (A pour émission et C pour la référence), or, une seule pouvait être envoyée sur le port série. J'ai donc poursuivi l'idée d'utiliser les capteurs indépendamment de la machine.

Image 4 : Niveaux logiques RS232 – TTL

On peut observer, sur ce graphique les différences de niveaux de tension entre RS232 et TTL.

Ici, on remarque qu'en RS232 un 0 est représenté par une tension de 13V, et un 1 par une tension de -13V



En revanche, le TTL ne fonctionne qu'entre 0 et 5V. Le 0 est représenté par 0V et le 1 par 5V

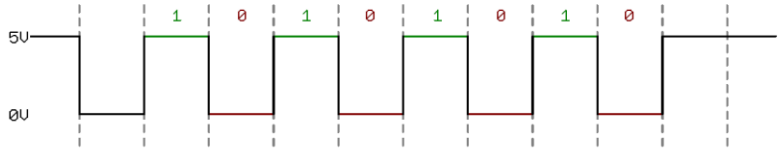
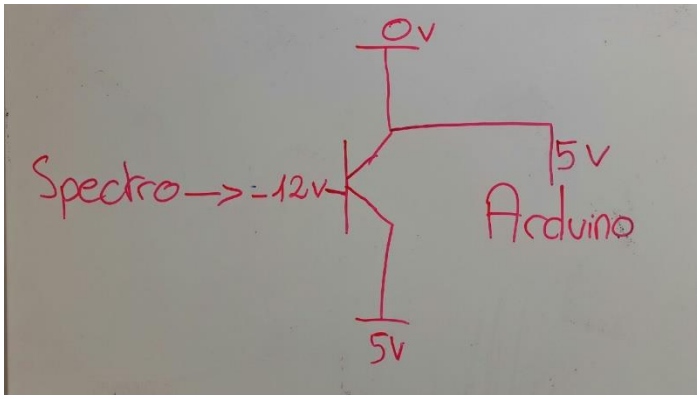


Image 5 : Conversion RS232 - TTL via transistor

Le principe de la conversion de niveau par un transistor se représente comme ci-dessous. Initialement, le transistor est un pseudo interrupteur en position ouverte. Le courant ne circule pas. Ici, dans la mesure où un 1 en RS232 vaut, moyennons, -12V et un 1 en TTL vaut 5V, il faut un transistor qui laisse passer le courant lorsqu'on a une valeur négative en provenance du spectrofluorimètre. Ici, il envoie -12V, ce qui rend le transistor conducteur et permet d'obtenir du 5V sur la carte Arduino. En revanche, dans le second cas, la tension est positive, le transistor n'est donc pas conducteur, donc la tension à la borne de l'Arduino vaut 0V.



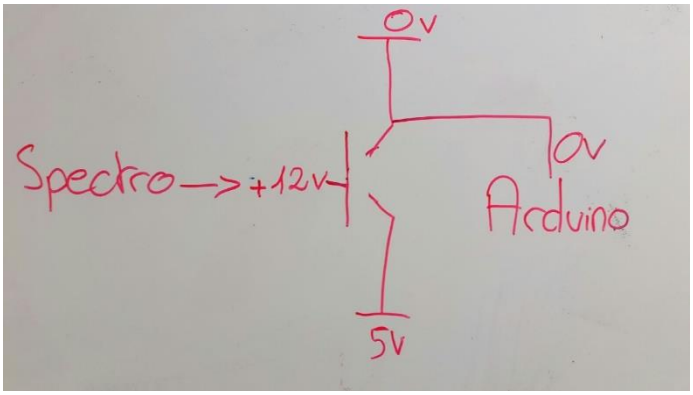
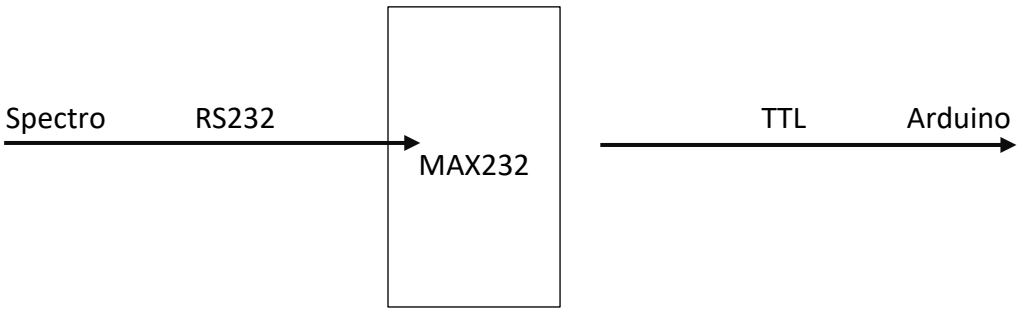


Image 6 : Conversion RS232 – TTL via MAX232

Ci-dessous, le principe de fonctionnement du MAX232, un simple convertisseur de niveaux, afin de comprendre les signaux entre spectrofluorimètre et Arduino.

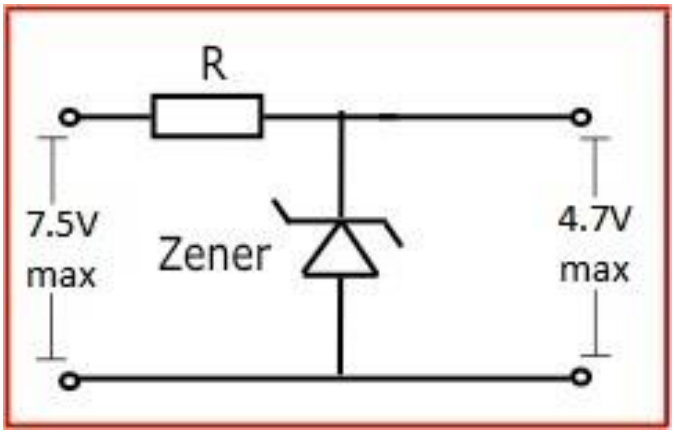


c. Communication Capteurs - Arduino :

Afin de pouvoir exploiter les signaux émis par photodiodes qui mesurent l'intensité lumineuse des voies A et C, j'ai dû comprendre leur fonctionnement. Ce sont de simples capteurs de lumières, qui renvoient une tension en fonction de l'intensité lumineuse qu'elles captent. Après une étude de la documentation et quelques tests, je me suis aperçu que la tension en sortie de ces capteurs variait entre 0V et 7,5V. Vulgairement, le 0V correspond au noir complet, à l'absence de lumière et le 7,5V correspond à une saturation, mais pas à un niveau de luminosité concret. En revanche, cette tension maximale n'est pas adéquate pour l'Arduino, pouvant mesurer des tensions de 5V tout au plus. J'ai donc tracé la courbe représentative de la tension en fonction de la luminosité, et il s'est avéré que la croissance était linéaire de 0 à 4,7V, mais dès ce seuil passé, elle devenait exponentielle. Cette dernière zone correspond donc à la plage de saturation du capteur. Disposant d'une tension variant de 0 à 4,7V et d'un microcontrôleur capable de mesurer une plage de 0 à 5V, il m'a suffi de trouver un moyen d'effacer ou de brider la saturation. J'ai réalisé un système de protection grâce à des diodes Zener. Une diode simple est un composant qui permet de ne laisser passer le courant que dans un sens, tout en réduisant la tension d'environ 0,8V. Une diode Zener, au lieu de simplement baisser la tension, possède une tension nominale, en dessous de laquelle le courant passe sans être altéré et au-dessus de laquelle la diode va consommer pour l'abaisser. Ce genre de composant est souvent utilisé dans des alimentations stabilisées, de sorte que la tension de sortie ne dépasse pas un seuil donné. J'ai trouvé une diode Zener de 4,7V (cf image 7), idéale donc pour le projet.

Image 7 : Diode Zener

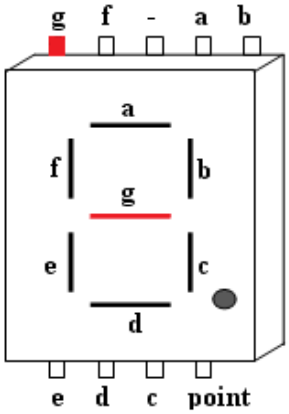
Utilisation de la Diode Zener ici, comme régulateur de tension. Toute tension inférieure à 4,7V ne sera pas modifiée, mais dans la zone de saturation, les 7,5V des capteurs seront altérés par la diode, afin de réguler une tension de 4,7V maximum.



Le spectrofluorimètre n'envoyant pas les longueurs d'ondes affichées, j'ai pensé à créer un capteur placé sur les afficheurs numériques, qui permettrait de récupérer le signal. Selon l'idée théorique de base, le principe était viable :

Un afficheur style « radio réveil » est constitué de 8 LEDs appelées segments. Un segment s'allume comme une lampe normale, lorsqu'on applique une tension sur la broche qui lui est reliée (cf image 8). Théoriquement, il semblait donc possible de connecter la carte Arduino sur un afficheur, de mesurer la tension sur chaque broche afin de savoir quels segments sont allumés ou éteints, et donc de connaître la valeur affichée.

Image 8 : Afficheur 7 segments



On peut constater que chaque broche est reliée à un segment. Le principe est donc de mesurer la tension sur chaque patte de l'afficheur, pour en déduire quels segments sont affichés. Ici, le rouge sur la broche g représente une tension appliquée. Le segment g s'allume donc.

En pratique il n'a pas été possible de réaliser un tel système car l'appareil dispose de 10 afficheurs, dont chaque segment est relié à ses homologues. En réalité, tous les afficheurs ne sont pas alimentés en même temps, le principe utilisé est le même qu'un film. C'est-à-dire qu'on allume un afficheur, on lui envoie la valeur à afficher, on l'éteint et on passe au suivant. On envoie des informations tellement rapidement que le cerveau ne peut toutes les analyser et elles lui semblent fluides. Finalement, la carte consommait trop de courant, et rendait le système d'affichage instable. J'ai donc abandonné l'idée de créer un capteur pour les longueurs d'ondes, au profit d'une solution logicielle ultérieure.

d. Communication Arduino – PC :

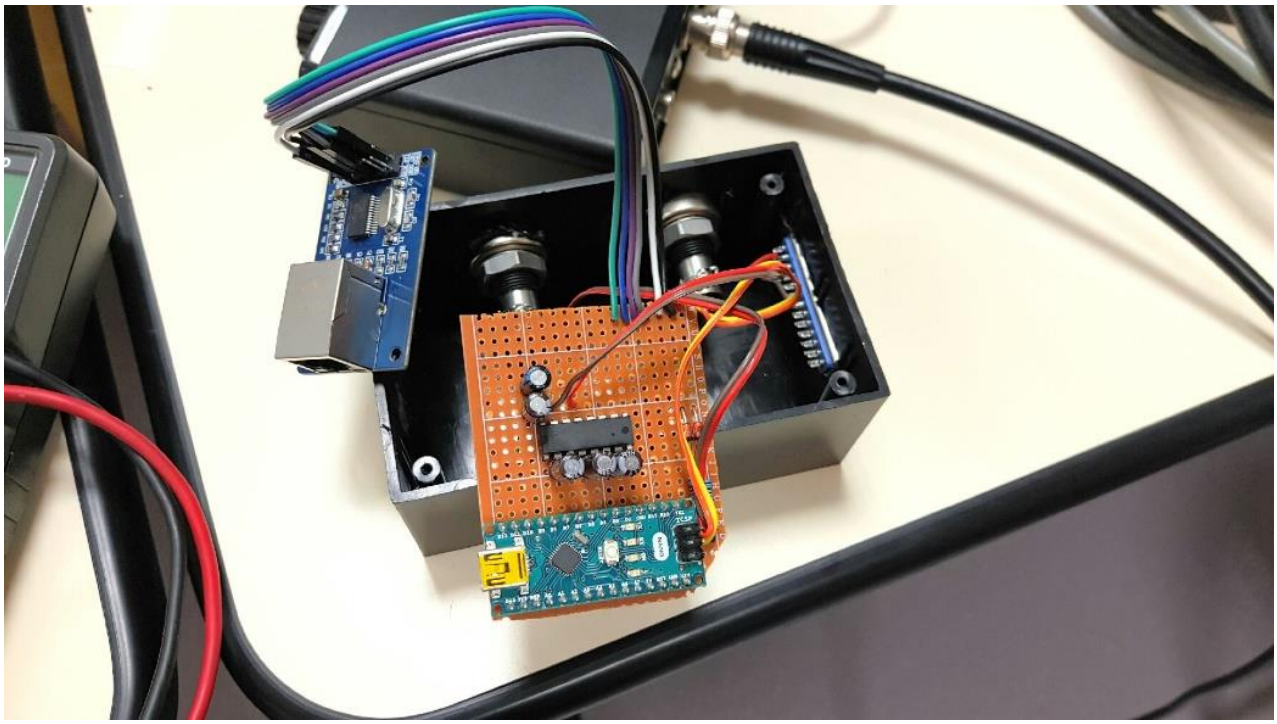
Une fois les composants choisis, deux options se sont présentées. Envoyer les informations simplement via le port USB de la carte Arduino sur le port USB d'un PC, ou créer un serveur Web sur microcontrôleur, en le reliant à internet via un adaptateur réseau spécifique. La première solution aurait donc consisté à la création d'une application, certainement C++, capable de lire les données du port série. La seconde constitue un défi technologique, liant un système électronique obsolète avec les principes technologiques actuels, de l'ère connectée. Cette seconde option paraissait plus intéressante en termes d'enrichissements personnels et éducatifs.

e. Réalisation d'un Circuit Imprimé

Afin de fournir un système documenté, j'ai créé le schéma électronique reliant tous les composants. J'ai donc profité de l'opportunité d'avoir les plans pour graver un circuit. Ne disposant pas du matériel nécessaire, j'ai tenté d'imprimer le plan sur papier glacé avec une imprimante laser. Cette association permet de profiter de la fine couche de plastique couvrant l'encre, qui l'empêche de « baver » lors d'un frottement trop rapide, mais également de pouvoir déposer cette couche sur un autre support par transfert de chaleur. Le papier glacé permet une bonne accroche, sans s'imbiber d'encre. Malheureusement, lors du transfert sur la plaque de cuivre du futur circuit imprimé, l'encre s'est étalée à plusieurs endroits. J'ai été contraint de stopper cette expérience car j'aurais exposé le projet à d'éventuels court-circuits, dans la mesure où plusieurs pistes se rejoignaient. J'ai donc finalement utilisé une plaque pré-percée, qui m'a permis de réaliser le circuit, un peu plus imposant en taille, mais pas moins efficace (cf image 9). J'ai également adapté un transformateur afin de pouvoir relier le boitier au secteur et non à un port USB.

Image 9 : Circuit Imprimé final

Sur cette image, on peut distinguer le microcontrôleur Arduino (en bas), le module Ethernet (à gauche, en bleu), et le convertisseur de niveaux logiques (au centre, rectangle noir). Le transformateur a volontairement été retiré pour des raisons de visibilité.

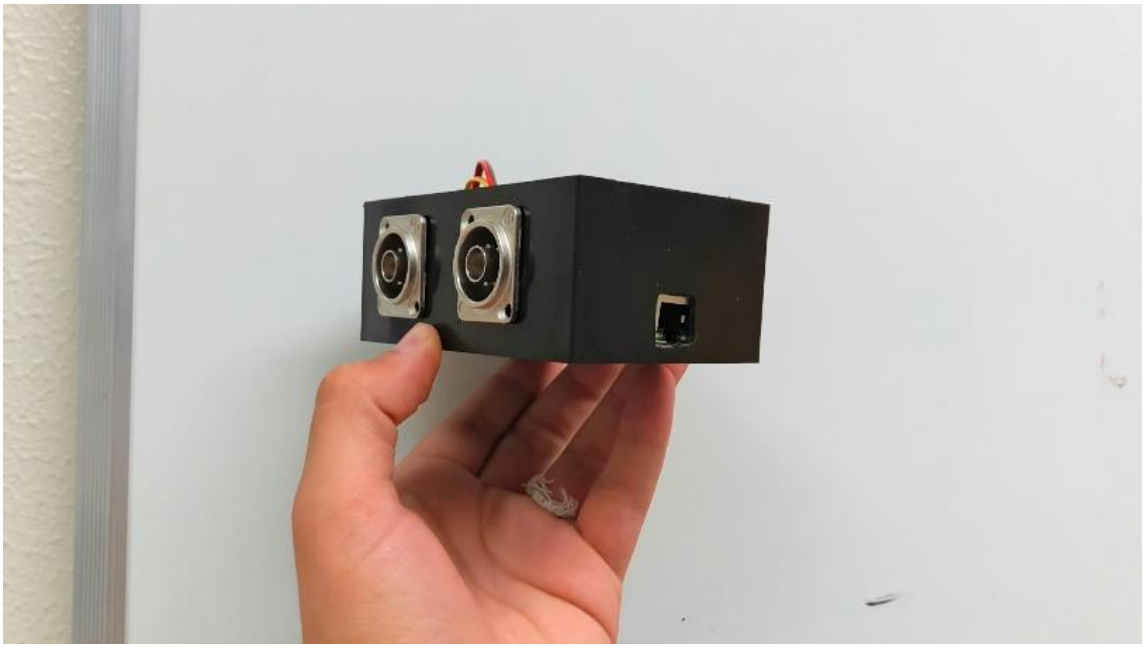


f. Conclusion I

Pour conclure cette première partie électronique, la liste des composants retenus est courte, et le prix de revient de ces derniers est très faible par rapport au prix d'une machine telle que celle étudiée. Bien qu'ayant rencontré quelques difficultés, ces dernières ont été surmontées principalement par de la recherche, de l'étude de documentation technique, et de tests. En résumé, le cœur de la partie électronique, le microcontrôleur Arduino communique avec le spectrofluorimètre via un convertisseur RS232←→TTL. Il récupère les informations des photodiodes avec un système de protection contre les surtensions et communique avec le réseau via un contrôleur Ethernet.

Image 10 : Boitier final

Sur le boitier terminé, on peut voir les deux fiches BNC permettant de relier les photodiodes ainsi le trou par lequel brancher le câble réseau.



3. Partie II : programmation embarquée

L'aspect embarqué de la programmation de cette partie vient du fait que le code est exécuté sur un microcontrôleur Arduino. Ce type de carte nécessite un code en C/C++ réduit. C'est-à-dire que tout code Arduino peut être utilisé sur un PC, mais tout code PC ne peut pas être utilisé sur Arduino. Cela vient de la mémoire réduite des microcontrôleurs. Un PC disposant de plusieurs Giga Octets de mémoire, vive ou morte, laisse plus de possibilités qu'une carte bridée à 2 Kilo Octets de mémoire.

a. Utilisation de bibliothèques

Pour parvenir à créer le programme embarqué, j'ai utilisé trois bibliothèques.

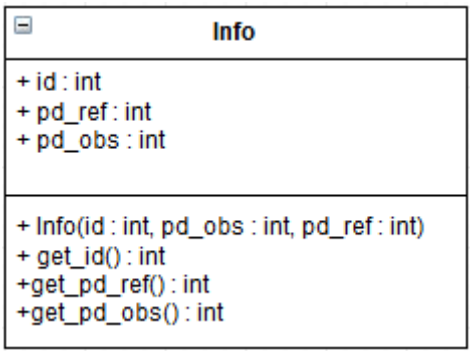
1. La première contenait le code nécessaire au bon fonctionnement du contrôleur Ethernet reliant l'Arduino au réseau. C'est grâce à elle que l'on peut, entre autre, savoir si un client émet une requête, et lui répondre.
2. La seconde permet de créer une liaison série virtuelle. C'est-à-dire que l'on crée une liaison série de deux broches (transmission et réception), permettant de communiquer avec le spectrofluorimètre. Elle nous défère la possibilité de savoir à quel moment le spectrofluorimètre envoie les données.
3. Enfin, la dernière, dénommée « Vector » a la même utilisation qu'en C. Elle permet de créer une liste ici, d'objets, sans limite de nombre. Contrairement à un tableau qui aurait une taille définie dans le code, un vecteur n'a pas de taille fixe, ce qui libère de la mémoire, dans le sens où cette dernière n'est pas occupée par un tableau potentiellement vide.

Toutes ces bibliothèques sont écrites en C++.

b. Création d'une classe

Afin de simplifier le code Arduino, j'ai développé une courte classe C++, permettant de créer des objets contenant un identifiant et deux valeurs correspondantes aux deux capteurs. Le vecteur décrit précédemment contient donc des objets de cette classe. Plus précisément, elle s'instancie avec trois entiers : un identifiant, une valeur pour la photodiode A et une valeur pour la photodiode C. Elle ne dispose comme méthodes hormis le constructeur, que d'un getteur par attribut (cf image 11).

Image 11 : Classe C++ développée



c. Création d'un serveur

Le serveur web crée ici est volontairement simpliste, afin d'éviter tout détournement de son utilisation primaire. Il démarre, se connecte au réseau avec une IP statique. Une fois opérationnel, il se met en écoute d'une requête en provenance d'un client. Le réseau de l'UBS, comme n'importe quel réseau d'entreprise est sécurisé par un filtrage d'adresse MAC. Si cette dernière est erronée, l'accès est interdit. Utiliser l'adresse MAC d'un autre appareil est considéré comme une usurpation d'identité. La seule solution légale à la connexion du microcontrôleur sur le réseau a donc été de voir avec la DSI afin qu'elle octroi le droit de connexion à l'Arduino sur une prise réseau.

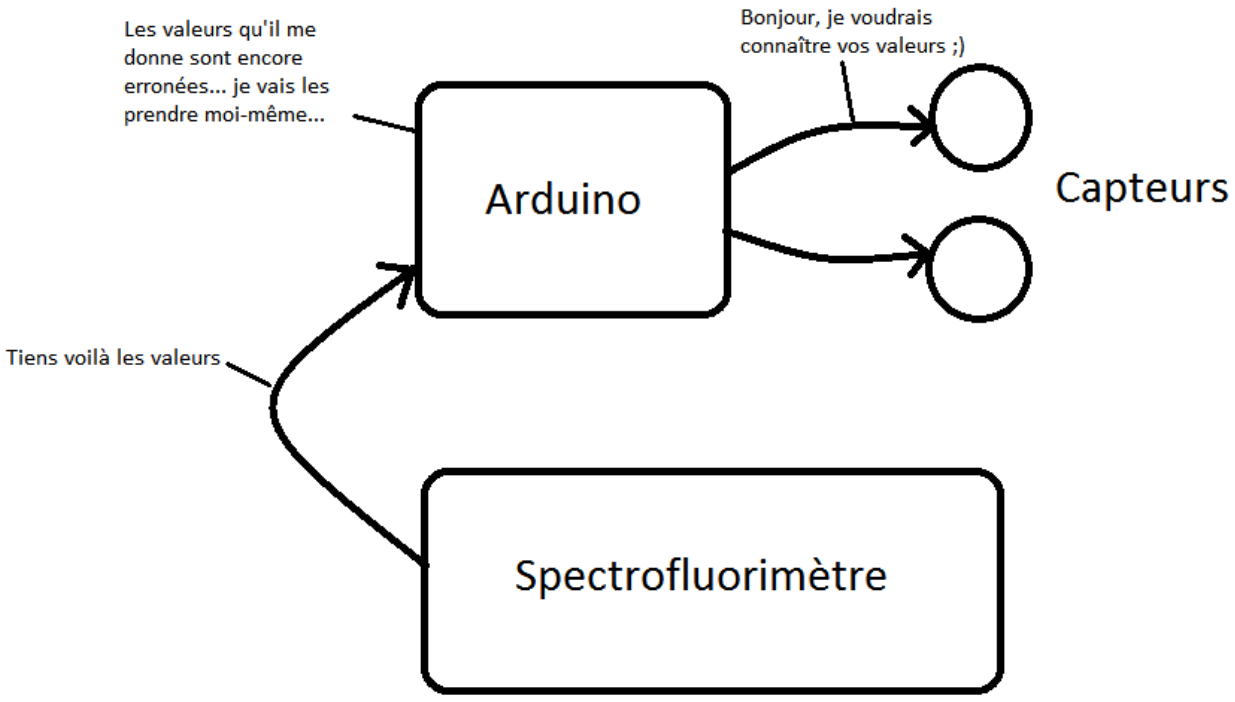
d. Réception et traitement des requêtes

Le Microcontrôleur est configuré pour répondre à deux types de requêtes. La première n'est qu'une simple lecture de tensions des capteurs à un instant T. La seconde est une demande de toutes les informations stockées en mémoire, avec les identifiants utilisés pour une correction d'erreur ultérieure. Dans les deux cas, les données sont transmises au format JSON. Dans la pratique, la première sert uniquement à indiquer les tensions à l'application. La seconde sert lors d'un scan, pour intégrer au graphique toutes les valeurs prises selon le tempo du spectrofluorimètre.

e. Lecture des capteurs

Le spectrofluorimètre n'envoyant pas les données souhaitées, il sert uniquement de temporisateur. C'est-à-dire que lorsqu'on reçoit des informations en provenance du port série, on ne les lit pas. On récupère les tensions des deux capteurs (valeurs moyennées sur 5 prises successives à 10ms d'intervalle ; cf image 12) pour les stocker dans une liste avec un identifiant.

Image 12 : Principe du code Arduino



f. Modes de connexion

L'application est prévue pour fonctionner selon deux modes (cf images 13-14). Pour le premier, l'Arduino est relié au PC directement via un câble Ethernet. Ceci permet de gagner en rapidité. Avec cette configuration, le temps de réponse de l'Arduino ne dépasse pas 100ms en utilisation intensive. Dans le second mode, on ajoute un serveur type entreprise. Toutes les requêtes passent donc par un serveur intermédiaire entre contrôleur et PC. L'inconvénient de cette configuration est la dépendance à un réseau fonctionnel. Lors du stage une mise à jour des serveurs a été effectuée. Le réseau n'était pas utilisable dans cette plage horaire. Il a donc été plus qu'utile de développer la configuration PC – Arduino rapidement.

Image 13 : Première configuration du module

Première configuration : Arduino ←→ PC

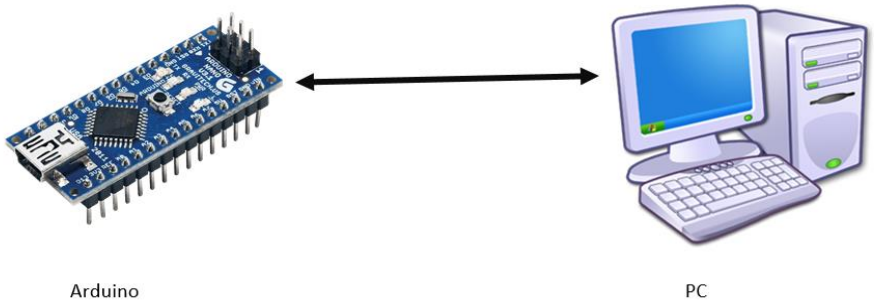
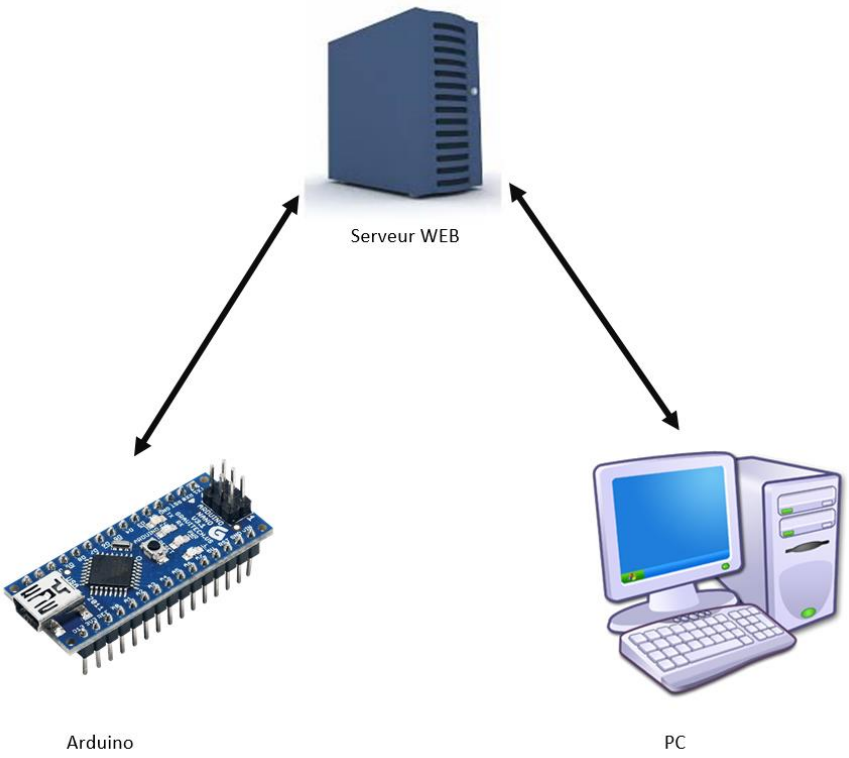


Image 14 : Seconde configuration du module

Seconde configuration : Arduino ←→ Serveur ←→ PC



g. Environnement de développement

Utilisant un microcontrôleur de marque Arduino (cf image 15), l'utilisation du compilateur de la marque était parfaitement adaptée. En revanche pour l'édition du code, j'ai préféré choisir un IDE indépendant et libre : Sublime Text (cf image 15). Le cahier des charges n'imposant aucune contrainte, j'ai fait le choix de ce logiciel de par mon habitude à développer avec.

Image 15 : Logos Arduino et Sublime Text

Ci-dessous les logos d'Arduino (récemment devenu Genuino en Europe), et de SublimeText

h. Conclusion II

Le développement de la partie programmation embarqué a demandé une longue lecture de classes, car la bibliothèque utilisée pour interagir entre la carte Arduino et le contrôleur réseau n'a pas de documentation officielle. Seul le code source est commenté. Cette étape a pris un certain temps, n'ayant jamais travaillé en réseau avec un microcontrôleur. Dans sa globalité, l'algorithme du programme embarqué reste relativement simple et court. D'une longueur d'environ 150 lignes, il est donné en intégralité en annexe et en version allégée ci-dessous (cf image 16). Il se contente d'un côté de recevoir une requête réseau via le contrôleur Ethernet, de mesurer deux tensions et d'envoyer une réponse. D'un second côté, il lit et enregistre les tensions lorsqu'il détecte une activité en provenance du spectrofluorimètre.

Image 16 : Code Arduino en version raccourcie

Ci-dessous, le code Arduino épuré des inclusions, variables, et du corps des méthodes secondaires. La première boucle, setup, est une méthode ne s'exécutant qu'une fois, au démarrage. Ici, elle sert uniquement à initialiser les liaisons séries avec le PC (pour du debug) et le spectrofluorimètre, ainsi qu'à démarrer le module Ethernet s'il est connecté.

La seconde boucle, loop, s'exécute à l'infini. Le fonctionnement est simple. Si des données proviennent du spectrofluorimètre, on relève les tensions des capteurs. Si on reçoit une requête réseau, on l'analyse pour en extraire les arguments et on traite en fonction.

```
void setup(){
  Serial.begin(38400);
  slm.begin(2400);
  if (ether.begin(sizeof Ethernet::buffer, mac) == 0){
      ether.staticSetup(ip);
}

void loop(){
  while(slm.available()){
    get_vals();
    break;
  }
  if (ether.packetLoop(ether.packetReceive())){
    bfill = ether.tcpOffset();
    char *data = (char *) Ethernet::buffer + pos;
    if(strncmp("GET /", data, 5) == 0){
      data+=5;
      if(strncmp("?onlyValues", data, 11) == 0)
        ether.httpServerReply(only_sensors());
      else if(strncmp("?scan", data, 5) == 0){
        ether.httpServerReply(send_values());
        v_infos=Vector<Info>();
      }
    }
  }
}

void get_vals(){
  ...
}

static word send_values(){
 ...
}

static word only_sensors(){
  ...
}
```

4. Partie III : programmation WEB

L'application WEB est une simple interface graphique pour le spectrofluorimètre. Elle est séparée en trois parties. La première, concerne les informations des capteurs et l'affichage des longueurs d'ondes actuelles. La seconde, le graphe en lui-même et la dernière, les boutons d'actions. L'utilisateur, au lancement, doit ouvrir un menu de configuration pour paramétrer le graphique. Une fois les informations entrées, il peut lancer l'acquisition. L'application va alors requêter l'Arduino pour obtenir les valeurs des capteurs et pouvoir tracer une courbe. Une fois l'acquisition terminée, l'utilisateur peut sauvegarder le graphique sous forme d'image, de fichier CSV ou encore JCAMP, pour permettre une étude ultérieure sur tableur.

a. Environnement de développement

Afin de créer cette application WEB, j'ai programmé en trois langages web : HTML, CSS et Javascript (cf image 17). Les deux premiers ne concernent que la partie visuelle de l'application, alors que le dernier en constitue le corps. Pour respecter le modèle MVC enseigné en cours, j'ai utilisé le framework Javascript Angular. Un module ainsi que deux contrôleurs ont donc été créés pour l'application. Un pour gérer l'affichage des valeurs actuelles et le second pour gérer les boutons et le graphique. Ce graphique est contenu dans un objet développé avec le framework Prototype, avec une bibliothèque déjà existante : ChartJs. Enfin, pour toute la partie esthétique j'ai utilisé un framework CSS, Materialize (cf image 18). Le principal avantage à l'utilisation d'un framework CSS est la réutilisation de code existant, assurant un rendu graphique plus propre qu'en réalisant tout manuellement.

Image 17 : Logo JavaScript - HTML - CSS

Ci-dessous, les logos, respectivement, du Javascript, HTML, et CSS

Image 18 : Logos Angular - Materialize - ChartJs

Ci-dessous, les logos d'Angular, Materialize et ChartJs

b. L'affichage des informations

Cette première partie est constituée d'un sélecteur radio (cf image 19), de trois barres de progression (cf image 20) et de deux zones de texte non modifiables (cf image 21). Le sélecteur permet de choisir le capteur à utiliser pour la création du graphique. Ce choix peut être l'un des deux capteurs ou leur ratio. Les trois barres de progression représentent les valeurs des deux capteurs et leur ratio. Elles sont créées par deux conteneurs l'un dans l'autre. Le conteneur arrière représente l'intervalle de progression et celui au premier plan représente la fluctuation. Dans la première version de l'application, ces barres progressaient de façon linéaire. Un problème était posé : les informations n'étaient pas pertinentes. En effet, l'Arduino renvoie une valeur entre 0 et 1024, correspondant à une tension entre 0 et 5V. Le cœur du problème résidait dans le fait, qu'en pratique, les valeurs renvoyées sont souvent faibles. L'échelle n'était donc pas réellement adaptée car à l'œil nu, il était impossible d'observer les variations de valeurs. Ceci gênait le réglage des capteurs avant de lancer un scan. Pour des questions de lisibilité, nous avons donc opté pour une échelle logarithmique, qui a la particularité de répartir les faibles valeurs sur un grand espace et les grandes valeurs sur un faible espace (cf image 20). Ce principe était plus adapté. En effet, il permet de voir une évolution de tension, même très faible. Enfin, les zones de texte ne sont pas modifiables par l'utilisateur, mais par le code Javascript. A la réception d'une information, on va incrémenter les longueurs d'ondes et les afficher.

Image 19 : Sélecteur Radio

Le message en dessous apparaît en laissant la souris 1,5 seconde sur le champ, sans cliquer, afin d'aider un utilisateur en éventuelle détresse. Ce dernier choisit le capteur / ratio à utiliser

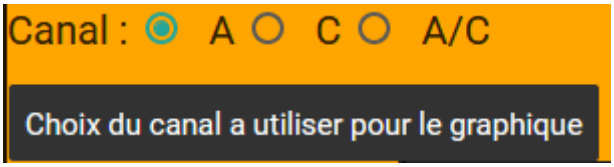


Image 20 : Barres de progression

Ci-dessous, la zone d'affichage des barres de progression. On distingue donc les barres représentatives des deux capteurs A et C, ainsi que le rapport A/C. L'échelle logarithmique est donc adaptée : à titre informatif, les valeurs renvoyées comprises entre 0 et 10 se situent entre le 0 et le 1. Celles comprises entre 10 et 100 sont réparties entre le 1 et le 2, etc…. Ici on peut voir que le canal C est proche de 1, donc la valeur envoyée par le capteur est proche de 10.

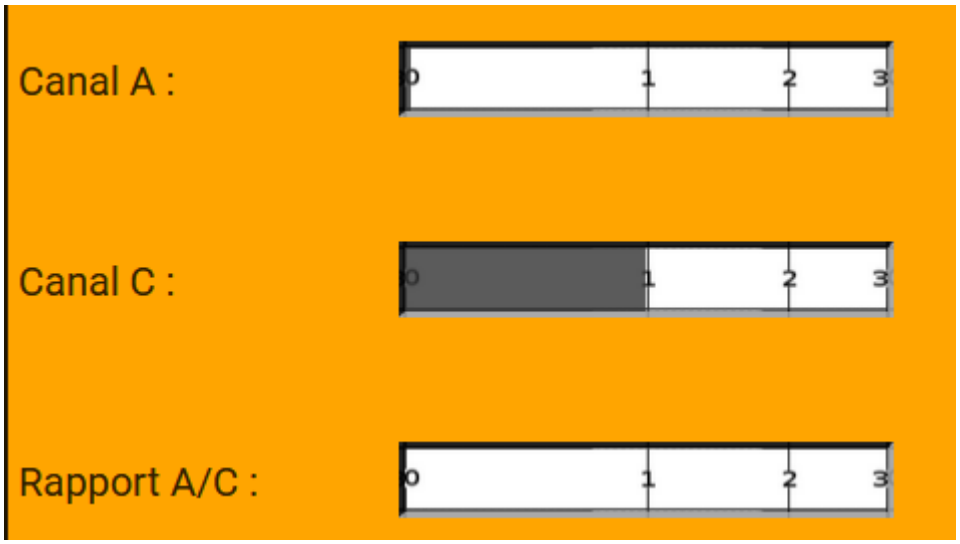


Image 21 : Affichage des longueurs d'ondes

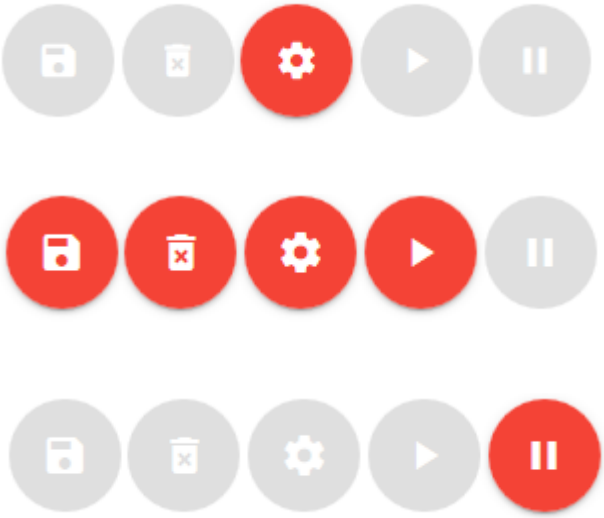
Ci-dessous, la zone d'affichage des longueurs d'ondes actuelles. Selon le type de graphe, l'une ou l'autre va s'incrémenter à chaque information reçue.

c. Boutons d'actions

Cette zone est répartie en 5 boutons : sauvegarde, remise à zéro, configuration, lancement et arrêt. Pour empêcher l'utilisateur d'effectuer une mauvaise manipulation, certains boutons sont grisés lorsqu'ils sont inutilisables. Ces boutons sont activés ou non en fonction de trois scénarios. Quand l'utilisateur arrive sur l'application, seul le bouton de configuration est cliquable. Une fois cette étape passée, tous sont cliquables, sauf le bouton « pause », qui lui est le seul à s'activer lorsqu'on lance l'acquisition par le bouton « play » (cf image 22).

Image 22 : Barre de boutons

Ci-dessous, la barre de boutons dans les trois configurations possibles. La première, lors du démarrage de l'application. La seconde, une fois la configuration terminée. Et la troisième, une fois l'acquisition lancée.



La partie sauvegarde fait un appel à une méthode définie dans l'objet du graphique. Cette méthode retourne une chaîne de caractère formatée en CSV ou en JCAMP, correspondant aux coordonnées de tous les points dans le graphique (cf image 23). Elle enregistre le résultat sous forme d'une URI et ajoute sur la page HTML un conteneur permettant de lancer le téléchargement du fichier. L'inconvénient du JCAMP, par rapport au CSV est l'impossibilité d'enregistrer plus d'une courbe. Nous avons donc choisi de ne sauvegarder que la moyenne dans ce format.

Image 23 : Fichier JCAMP-DX

Ci-dessous, la même série de valeurs sauvegardée en JCAMP et en CSV.

Ici, le JCAMP-DX. On remarque un certain nombre de lignes avant les valeurs. Ces lignes servent à configurer le logiciel qui lira le fichier, afin qu'il calibre le graphique. En revanche, une seule série est enregistrée sous la forme de couples X, Y.

```
##TITLE= spectrofluorescence
##JCAMP-DX= 5.00
##OWNER= UBS
##DATA TYPE= FLORESCENCE SPECTRUM
##DATA CLASS= PEAK TABLE
##DATE=4.5.17
##TIME=12:38:11
##ORIGIN= Université Bretagne Sud, salle F091
##XUNITS=Nanometers
##YUNITS=Relative Luminescence
##FIRSTX=2
##LASTX=12
##NPOINTS=6
##XFACTOR=1
##YFACTOR=1
##XYDATA= (XY..XY)
2,70;
4,260;
6,69;
8,96;
10,964;
12,967;
##END=
```

En revanche, dans le format CSV, on peut observer que le superflu n'existe pas. La forme tableau en ressort plus facilement. En première ligne, sont présentes, les dénominations de chaque colonne. Chaque ligne suivante représente une ligne du tableau.

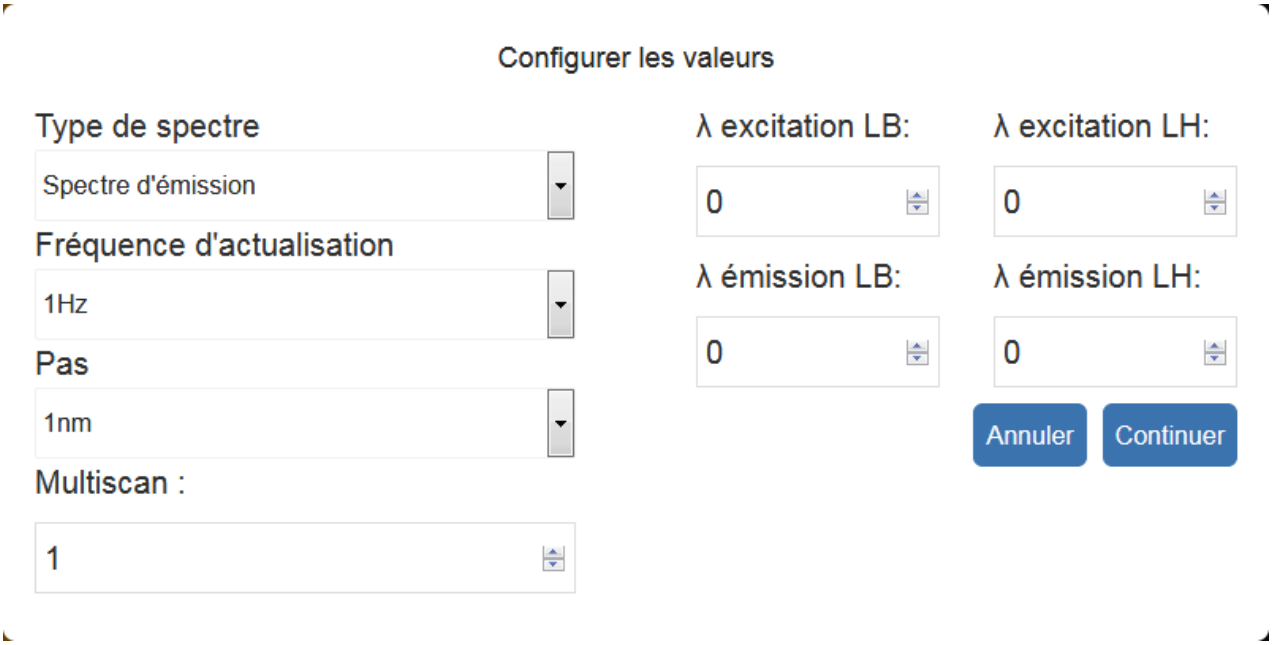
```
Longueur d'onde; scan 1; scan2
2;      70;     32
4;      260;    25
6;      69;     687
8;      96;     254
10;     964;    265
12;     967;    1
```

Lors de la remise à zéro, on efface simplement tous les points présents dans le graphique en remplaçant le tableau les contenant par un tableau vide. Ensuite on remet les longueurs d'ondes affichées à leurs valeurs initiales.

Pour la configuration, j'ai utilisé une bibliothèque existante, permettant d'ouvrir des pop-ups esthétiques. J'ai dû modifier cette dernière, ne permettant, de base, que l'affichage de champs de texte. J'ai donc ajouté la possibilité de créer un champ multichoix et des champs numériques avec des bornes inférieure et supérieure (cf image 24). Une fois les informations entrées, le graphe est créé.

Image 24 : Pop-Up

Ci-dessous la pop-up de configuration de l'application. Chaque information est vérifiée avant d'être envoyée dans le programme



Enfin, les boutons « play » et « pause » font appel à des méthodes qui vont démarrer ou arrêter un « Interval ». Un « Interval » représente un appel à une méthode toutes les x secondes. Un premier est lancé au démarrage de l'application afin de récupérer les valeurs instantanées des capteurs. Lors du clic sur le bouton « play », ce premier est stoppé et laisse place à un second afin de récupérer les valeurs pour le graphique. Le clic sur le bouton « pause » fait effet inverse.

d. L'affichage du graphique

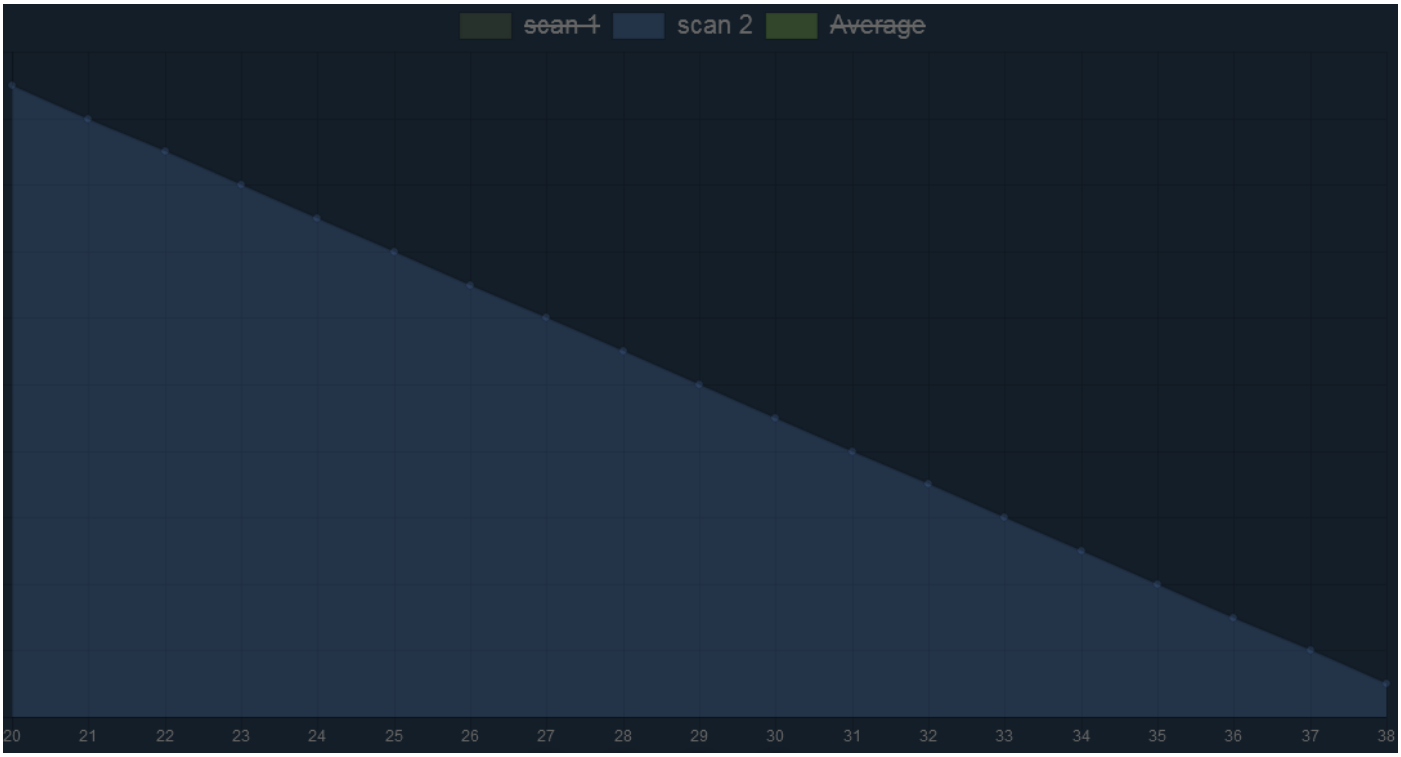
Une fois le menu de configuration complété, le graphique est créé et affiché vide à l'écran. La majeure partie du travail n'a pas été le graphique en lui-même, mais l'étude de toute la documentation d'utilisation, afin de pouvoir le paramétrer.

La partie du graphique concernant les données à afficher peut se résumer par un tableau pour l'axe des abscisses, un nom éventuel et des sets de données correspondants aux coordonnées des ordonnées. Chaque set de données contient un nom, une couleur pour sa courbe et un tableau équivalant à une liste de coordonnées Y. Chaque set possède son tableau de Y, en revanche le tableau de X est commun à tous.

Dans la première version, une seule acquisition était réalisée. Pour l'évolution vers un système plus fonctionnel, il a été nécessaire de trouver un moyen d'afficher plusieurs acquisitions. Deux options se sont présentées : afficher autant de graphes que de scans sur ou afficher autant de courbes que de scans sur un seul graphe. Cette dernière option semblait plus simple au niveau de la lisibilité de l'interface par l'utilisateur. Chaque scan de la machine est représenté par un set de données. Une fois une courbe affichée, on peut choisir de la cacher en cliquant simplement sur son label. Ce dernier se raye et la courbe disparaît de l'écran, mais n'est pas supprimée (cf image 25).

Image 25 : Multi-Graphe désélectionné

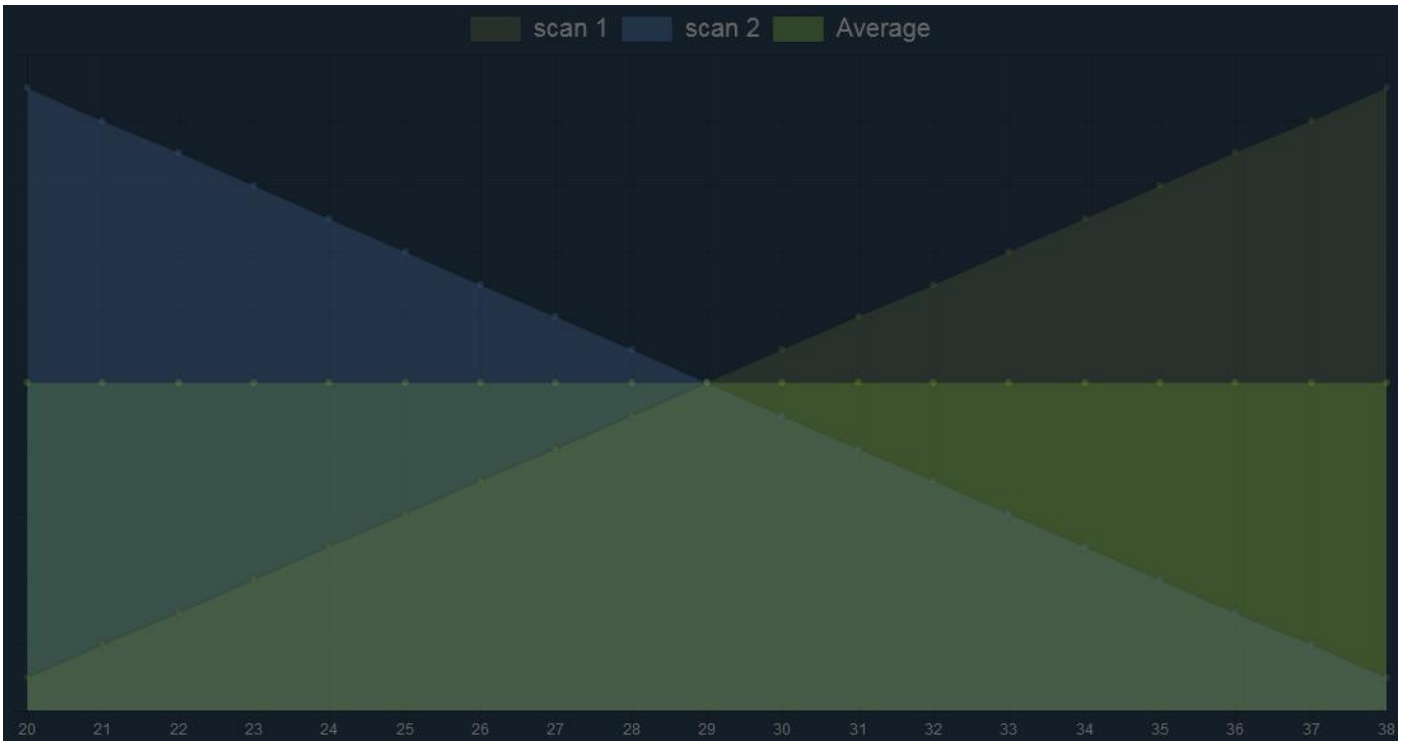
Ici, on peut voir qu'il y a en tout deux scans, plus la moyenne. Le premier scan et la moyenne ont été cliqués, ils sont donc rayés et ne s'affichent pas



J'ai choisi de toujours créer un set de plus qu'il n'est nécessaire afin de réaliser une moyenne des courbes affichées. C'est-à-dire que lors du calcul de cette dernière, on vérifie pour chaque set de données s'il est affiché ou non. Cette moyenne est recalculée automatiquement lorsqu'on affiche / cache une courbe ou lorsqu'un nouveau point est ajouté dans le graphe (cf image 26).

Image 26 : Multi-Graphe complet

Voici la même image que précédemment, mais cette fois-ci toutes les courbes sont affichées.



En testant l'application, nous nous sommes rendus compte que le graphique, lorsqu'on entrait un grand nombre de valeurs, était illisible au niveau des légendes de l'axe X. En revanche, j'ai remarqué que toutes les valeurs n'étaient pas affichées. Il a donc fallu trouver dans le code, à quel endroit ce « pas de saut de valeur » était calculé. Après quelques recherches, il s'est avéré que ce pas était calculé, entre autres, en fonction de la taille du graphique dans la fenêtre et du nombre de points. J'ai donc multiplié le rapport afin d'obtenir un pas plus élevé (cf image 27).

Image 27 : Affichage de l'axe des Abscisse

Ci-dessous, l'axe X avant et après modification de la bibliothèque. Dans les deux cas, les valeurs vont de 0 à 999.

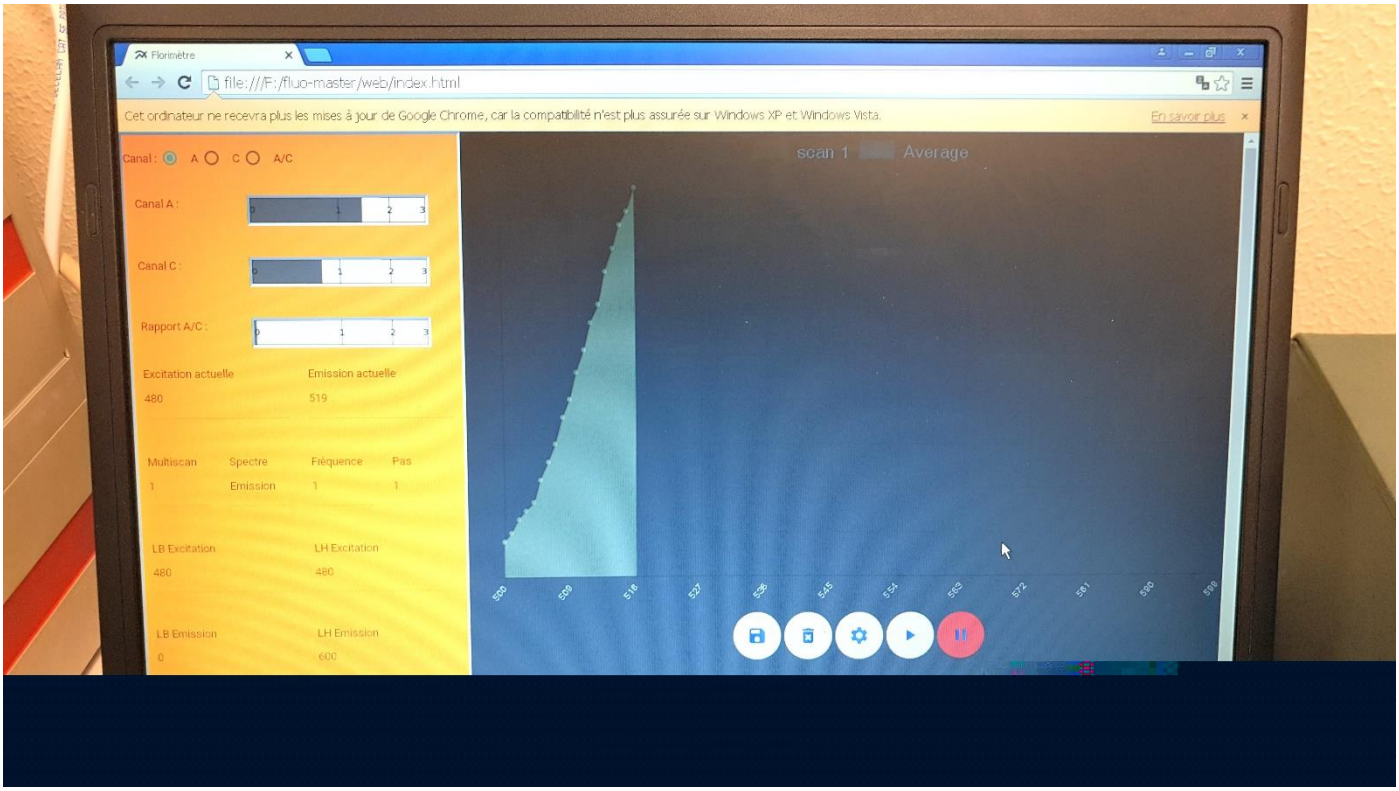
e. Émission de requêtes

Tout comme l'Arduino ne peut interpréter que deux types de requêtes, l'application ne peut en envoyer que deux. Sur cette partie, l'utilisation de la bibliothèque Ajax a permis de simplifier l'écriture et la compréhension du code. Les deux requêtes sont semblables, seuls varient l'URL, qui détermine si l'on veut les informations actuelles ou toutes celles enregistrées, et la fonction de callback. Cette fonction sert de retour, c'est-à-dire que lorsqu'on reçoit une réponse en provenance de l'Arduino, on la traite en exécutant ce callback. L'une des principales difficultés a été le « Cross Origin ». C'est une sécurité présente sur tous les navigateurs web actuels (sauf IE), qui interdit à une page web de questionner un serveur différent que celui sur lequel elle a été chargée. La requête partait bien, était reçue par l'Arduino, mais la réponse était bloquée par le navigateur web. Le contournement de cette sécurité a été possible en autorisant cette technique dans la réponse de l'Arduino à la requête du PC.

f. Conclusion III

Cette partie programmation WEB a occupé une place prépondérante dans la répartition du temps de travail. La base s'est réalisée rapidement, mais il a fallu sans cesse revenir sur le code crée afin d'ajouter de nouvelles sécurités, corriger des bugs, ou tout simplement repenser les fonctionnalités. La lecture de la documentation de la partie graphe a demandé beaucoup de temps. L'affichage en lui-même est simple, mais toute la partie configuration du graphique est répertoriée par ordre alphabétique, comme le clic sur les labels pour cacher une courbe. Ne connaissant pas le nom des options recherchées, il a fallu lire la documentation dans son intégralité. Enfin, une fois l'application terminée, j'ai rédigé des procédures d'installation et d'utilisation du matériel.

Image 28 : Visuel complet de l'application



Conclusion générale

En comparaison avec les exigences demandées dans le cahier des charges, le projet est terminé. L'interface de base demandée est fonctionnelle, permet l'acquisition de données, la sauvegarde de ces dernières, et la possibilité de voir l'évolution des capteurs en temps réel (diagramme des cas d'utilisation final en Annexe 2).

Bien que terminé, le projet est perfectible et ne pourra cesser de l'être. Par exemple, sur la partie logicielle, l'interface pourrait être recréée de façon plus esthétique, ou tout simplement intégrer de nouvelles fonctionnalités non pensées au début. En revanche, sur la partie électronique, la seule limite pour améliorer le projet serait l'imagination. L'avantage des microcontrôleurs tels que celui utilisé est leur grande compatibilité. Ainsi, il serait techniquement possible de créer une application smartphone fonctionnant en Bluetooth, ou bien n'importe quel système de communication sans fil.

J'ai eu la chance de réaliser mon stage dans un domaine qui me passionne, selon un mode de travaille me correspondant totalement. J'ai pu découvrir de nouvelles technologies et protocoles de communication, ainsi qu'une machine sur laquelle, sans ce stage, je n'aurais sans doute jamais travaillé. Bien que travaillant dans une salle à l'écart des bureaux des enseignants chercheurs, ils ne m'ont pas mis à l'écart pour autant.

La tâche qui m'a été confié m'a donné une totale autonomie sur la durée du stage. Cette dernière a instauré une certaine confiance entre mon tuteur et moi. Je pense avoir réussi à développer une solution électronique / informatique répondant aux spécifications imposées, tant sur la partie solution, que sur le prix de revient des composants de cette dernière.

J'ai apprécié le fait de travailler seul sur ce projet, me permettant d'utiliser ma méthode de travail sans contraindre un membre d'une équipe.

Je conclurais en ajoutant que j'ai conscience d'avoir eu la chance de pouvoir effectuer mon stage dans une bonne ambiance, dans un domaine qui me plaît, avec des personnes chaleureuses que je remercie de m'avoir accueilli.

Glossaire

Ajax : Bibliothèque combinant l'utilisation de requêtes, et la modification de la structure HTML.
Angular : Framework simplifiant l'utilisation de certains principes, comme le modèle MVC.
Arduino : Marque de microcontrôleurs.
BASIC : Langage de programmation de haut niveau, simplifiant l'utilisation.
Bibliothèque : Code réutilisable (peut être contenue dans un Framework).
Callback : Méthode appelée au retour d'une requête.
ChartJs : Bibliothèque Permettant d'utiliser un graphe.
Contrôleur Ethernet : Carte permettant d'ajouter une prise Ethernet à tout matériel compatible.
CSS : Feuille de style pour site web (ex : polices, tailles de texte, couleur…).
CSV : Format de fichier utilisé dans les systèmes de sauvegarde et exploitable via tableur.
Difracter une source lumineuse : « éclater » cette source afin d'en obtenir le spectre.
Diode : Composant électronique protecteur (Un seul sens de passage pour le courant).
Diode Zener : identique à une diode, mais protège également des surtensions.
Émission : ici, représente la fluorescence de l'échantillon et le Monochromateur la traitant.
Excitation : Ici, représente la source lumineuse et le Monochromateur irradiant un échantillon.
Framework : Un ensemble de composants réutilisables allégeant le code d'une application
IP : Adresse logicielle d'un PC (en comparaison, surnom d'un individu : il peut changer).
Javascript : Langage de programmation WEB back-end.
JCAMP : Format de sauvegarde utilisé dans le domaine de la recherche.
MAC : Adresse physique d'un PC (en comparaison, nom d'un individu : il ne peut changer).
Materialize : Framework CSS simplifiant la partie esthétique.
Microcontrôleur : Mini-ordinateur réalisant de simples opérations.
Monochromateur : Réseau de diffraction de lumière monté sur un moteur pas à pas
MVC : principe de programmation différenciant back-end, front-end, et contrôleur, reliant les deux.
Photodiode : Capteur de luminosité.
Référence : Ici, Photodiode utilisée pour normaliser la valeur observée.
Réseau de diffraction : Composant permettant de diffracter une source.
RS232 : Protocole de communication de trames binaires (-12V : 1 ; 12V : 0).
RS232 : Protocole de communication de trames binaires (5V : 1 ; 0V : 0).
URI : Chaîne de caractères identifiant une ressource non existante sur un réseau.
URL : Chaine de caractères identifiant une ressource existante sur un réseau.

Table des figures

Résumé

La formation DUT Informatique se terminant par une période de 10 semaines de stage en entreprise, j'ai effectué ce dernier au sein de l'Université Bretagne Sud, plus jeune université de Bretagne.

J'ai pu réaliser en autonomie une interface pour un ancien spectrofluorimètre. Ce projet avait pour but la réhabilitation de ce dernier dans une salle de TP. J'ai donc mêlé électronique, informatique embarquée et programmation WEB afin de parvenir au résultat escompté.

Ce stage m'a initié à de nouvelles technologies, de nouveaux protocoles et techniques de communication.

Summary

Computer Science Diploma ends with a ten weeks internship in a company. I did mine in Université Bretagne Sud, youngest university in Bretagne.

I was able to carry out autonomously an interface for an old spectrofluorimeter. This project was aimed at rehabilitating the latter in a laboratory. So I mixed electronics, embedded computing and WEB programming in order to achieve expected result.

This internship introduced me to new technologies, protocols and communication techniques.

Annexes

Annexe 1 : Code Arduino en intégralité

```
#include <EtherCard.h>
#include <SoftwareSerial.h>
#include <Vector.h>
#include "Info.h"
//Debug send informations on Serial Port
#define DEBUG true
#define PD_A A0
#define PD_C A1
// ethernet mac address and ip
//static byte mac[] = { 0xB4,0xB5,0x2F,0xC6,0x7B,0x6D};
static byte mac[] = { 0x48,0x49,0x54,0x4C,0x45,0x52 };
//static byte ip[] = { 192,168,0,2};
static byte ip[] = { 172,17,96,72};
unsigned long int id=-1;

//Buffer for http response
byte Ethernet::buffer[500];
BufferFiller bfill;

//Declaration of serial communication with MAX232
SoftwareSerial slm(8,9);
//Vector for save datas between two requests
Vector<Info> v_infos;

void setup(){
  Serial.begin(38400);
  slm.begin(2400);
  #ifdef DEBUG
    Serial.println(F("démarrage"));
  #endif
  //if ethernet module is disconnected, alert on serial port
  if (ether.begin(sizeof Ethernet::buffer, mac) == 0){
    #ifdef DEBUG
        Serial.println(F("Failed to access Ethernet controller"));
    #endif
    }
    //if (!ether.dhcpSetup()){
     // Serial.println("DHCP failed");
      ether.staticSetup(ip);
    //}
  //Set up network connection
  //ether.staticSetup(ip);
  //Print IP adress on serial
  Serial.print(ether.myip[0]);
  Serial.print(".");
  Serial.print(ether.myip[1]);
  Serial.print(".");
  Serial.print(ether.myip[2]);
  Serial.print(".");
  Serial.println(ether.myip[3]);
  #ifdef DEBUG
    Serial.println(F("Prêt à fonctionner"));
  #endif
}


void loop(){
  //If we have datas from the SLM, get actual values
  while(slm.available()){
    slm.readString();
    get_vals();
    break;
  }
  //If we have a request, sent a response and erase datas
  word pos = ether.packetLoop(ether.packetReceive());
  if (pos){
    delay(1);
    bfill = ether.tcpOffset();
    char *data = (char *) Ethernet::buffer + pos;
    if(strncmp("GET /", data, 5) == 0){
      data+=5;
      //if get attribute represent a simple sensor read request
      if(strncmp("?onlyValues", data, 11) == 0)
        //send only actual sensor values
        ether.httpServerReply(only_sensors());
      //else if get attribute represent a scan request
      else if(strncmp("?scan", data, 5) == 0){
        //send all sensor values
        ether.httpServerReply(send_values());
        v_infos=Vector<Info>();
      }
    }
  }
  delay(50);
}

/**
 *  Calculate an average for A and C from 5 values for each
 */
void get_vals(){
  int a = 0;
  int c = 0;
  for(int i = 0; i< 5; i++){
    a += analogRead(PD_A);
    c += analogRead(PD_C);
    delay(10);
  }
  Serial.println(id);
  v_infos.push_back(Info(id++ ,(int)c/5, (int)a/5));
```

```
  #ifdef DEBUG
    Serial.println(F("Récupération des infos"));
  #endif
}

/**
 *  return a formatted JSON response, with all points in the list
 *  @return     JSON response in buffer
 */
static word send_values(){
  bfill = ether.tcpOffset();
  bfill.emit_p(PSTR("HTTP/1.1 200 OK\r\n"
                    "Content-Type: text/json;charset=utf-8\r\n"
                    "Server: Arduino\r\n"
                    "Access-Control-Allow-Origin: *\r\n"
                    "Connection: close\r\n"
                    "\r\n"
                    "["));
  //for each data in vector, add value to the response
  if (v_infos.size() > 0)
    bfill.emit_p(PSTR("{\"id\":$D,\"a\":$D, \"c\":$D}"),v_infos[0].get_id(),v_infos[0].get_pd_obs(), v_infos[0].get_pd_ref());
  for(int i = 1; i < v_infos.size(); i++)
    bfill.emit_p(PSTR(",{\"id\":$D,\"a\":$D, \"c\":$D}"),v_infos[i].get_id(), v_infos[i].get_pd_obs(), v_infos[i].get_pd_ref());
  bfill.emit_p(PSTR("]"));
  return bfill.position();
}
/**
 * Return a formatted JSON response, with only actual sensor values
 * @return      JSON response in buffer
 */
static word only_sensors(){
  int a = 0;
  int c = 0;
  for(int i = 0; i< 5; i++){
    a += analogRead(PD_A);
    c += analogRead(PD_C);
    delay(10);
  }
  a/=5;
  c/=5;
  bfill = ether.tcpOffset();
  bfill.emit_p(PSTR("HTTP/1.1 200 OK\r\n"
                    "Content-Type: text/json;charset=utf-8\r\n"
                    "Server: Arduino\r\n"
                    "Access-Control-Allow-Origin: *\r\n"
                    "Connection: close\r\n"
                    "\r\n"
                    "callback({\"a\":$D, \"c\":$D});"), a, c);
 return bfill.position();
}
```

Annexe 2 : Diagramme des cas d'utilisation