IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département Informatique

Développement d'un logiciel de cartographie d'irradiance, modernisation de l'acquisition de données sur spectrophotomètre Lambda 9 et conversion de fichier de mesure d'un goniophotomètre

Rapport de stage de BUT Informatique
Entreprise : DR. FISCHER Europe S.A.S.

Gaëtan PINOT
Maître de stage : Charles RAMMAERT
Référent de stage à l'IUT : Pierre-André GUÉNÉGO
Année universitaire 2023–2024

Remerciements

Je tiens à remercier Charles RAMMAERT, mon maître de stage, pour m'avoir accueilli dans son service et m'avoir donné l'opportunité de travailler sur des projets intéressants et variés.
Je remercie aussi les autres personnes de l'entreprise avec qui j'ai été en contact pour leur accueil.
Je remercie également Pierre-André GUÉNÉGO, mon référent de stage à l'IUT, pour m'avoir indiqué ses attentes pour le rapport et la soutenance.
Je remercie mes parents pour leur soutien, mon père pour m'avoir donné le contact de Charles.

Table des matières

1 Introduction

J'ai effectué mon stage de 8 semaines au sein du service qualité de l'entreprise DR. FISCHER Europe S.A.S., un fabriquant de lampe. Pendant ce stage, j'ai programmé un logiciel permettant de contrôler des appareils de mesure afin d'effectuer des mesures d'irradiances. J'ai remplacé l'imprimante d'un appareil de mesure par un ordinateur pour moderniser l'acquisition de donnée. De plus j'ai programmé un logiciel de conversion de fichier d'un instrument de mesure vers un logiciel spécifique.

1.1 Présentation de l'entreprise

DR. FISCHER Europe est une entreprise de développement et de fabrication de lampes et de systèmes, basée à Pont-à-Mousson depuis 1982. Anciennement une usine Philips, elle a été rachetée par DR. FISCHER Group, une entreprise allemande. Avec une équipe de 135 personnes, elle opère dans le domaine depuis plusieurs décennies. En 2023, l'entreprise a réalisé un chiffre d'affaires de 12 millions d'euros.
Cette usine produit une gamme variée de lampes, notamment des lampes infrarouges, UV, halogènes, LED, et dans certains cas, le système complet du luminaire incluant les réflecteurs. Ces lampes sont utilisées dans diverses applications, telles que le chauffage des terrasses, le séchage de la peinture ou des encres, ou même dans la signalisation des voies ferrées.
Le département Qualité, dans lequel j'ai effectué mon stage, a notamment pour objectif de garantir la qualité des lampes produites.

1.2 Environnement de travail

Le service Qualité est composé de 7 personnes. Son rôle est de s'assurer que les lampes envoyées aux clients sont conformes à la fiche technique de la lampe. Par exemple, il faut que la lampe délivre la bonne puissance, qu'elle dure le nombre d'heures nécessaire et qu'elle ne mette pas en danger l'utilisateur. Cela est notamment assuré par des contrôles et des mesures de certaines lampes en sortie de production.

J'ai un poste de travail fixe, dans le bureau de mon maître de stage avec un ordinateur. Cependant, comme je travaille avec des logiciel sous licences qui ne sont pas présent sur tous les ordinateurs et sur des équipements dont je ne peux attester le fonctionnement que si je suis à coté, je suis amené à ne pas toujours travailler sur le même poste.
Mes horaires de travail sont souples, je peux arriver entre 7h et 9h et repartir entre 15h45 et 18h, tant que je fait 7h par jour. J'ai effectué ce stage dans le département Qualité et pas Informatique car le rôle du département Informatique est de s'occuper de l'infrastructure informatique de l'entreprise, en tant que développeur je n'aurai pas eu ma place dans ce département.

1.3 Missions

Dans le cadre de sa mission, le département Qualité fait des prélèvements de lampes en production et les teste sur différents aspects, notamment l'irradiance* de ces lampes. Ces tests sont réalisés par le banc de mesure XY. (Voir figure 1, page 7).
Ce banc mesure l'irradiance de la lampe sur un quadrillage, ou une croix de points. Il fait cela en déplaçant une cellule de mesure sur un axe X et un axe Y. Les deux axes sont pilotés par des moteurs pas à pas, qui sont branchés à un contrôleur. Le contrôleur est connecté à un ordinateur, sur lequel un programme LabVIEW, interne à l'entreprise, envoie les commandes de déplacement. Il y a plusieurs cellules de mesure, pour différentes plages d'irradiance.

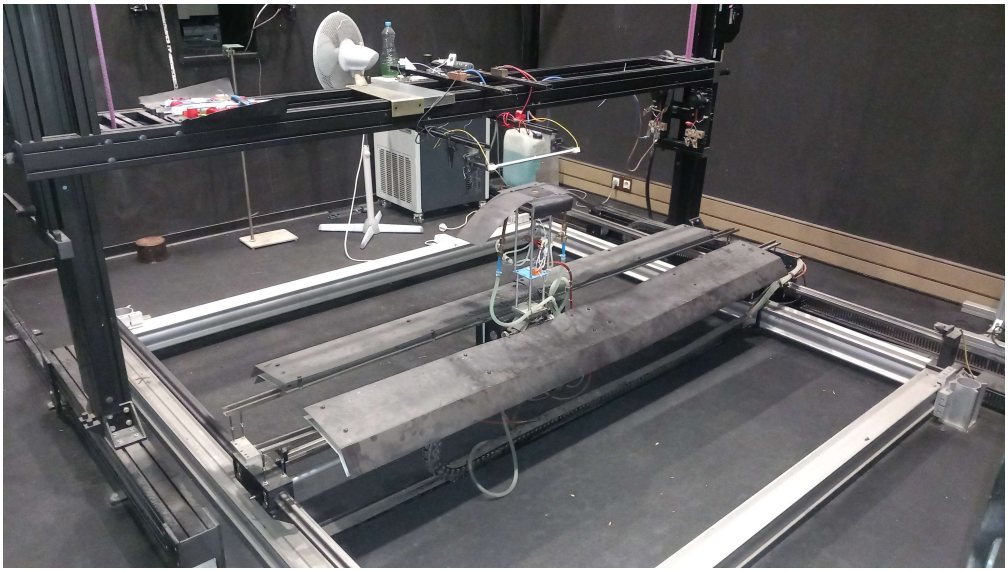


FIGURE 1 – Photo du banc XY

En 2022, le contrôleur du banc a cessé de fonctionner. Il est remplacé par un nouveau contrôleur qui n'est plus compatible avec l'ancien programme LabVIEW.
L'opérateur devait alors entrer manuellement les coordonnées dans le contrôleur et lire les valeurs de l'irradiance sur l'ordinateur pour chaque point du quadrillage. Cela prenait beaucoup de temps, ce qui pousse les demandes à être minimisées. Ma mission est de faire fonctionner à nouveau le banc de mesure XY.

Ma deuxième mission principale est liés au Lambda 9. Le Lambda 9, de Perkin-Elmer, est un spectrophotomètre qui permet de mesurer des valeurs de transmission, de réflexion et d'absorption de différents matériaux dans le domaine de l'ultraviolet, du visible et de l'infrarouge. Il est utilisé, par exemple, par le département Développement pour vérifier les longueurs d'ondes que certains matériaux absorbent le mieux, afin de suggérer au client la lampe la plus approprié à l'application, ou alors quel matériau est le meilleur réflecteur pour ces longueurs d'ondes. C'est une machine datant des années 80, elle imprime la courbe de données par une imprimante thermique spécifique au Lambda 9. La pointe chauffante de l'imprimante est cassée, et il est impossible de trouver une pièce de rechange.
La solution temporaire est d'attacher un stylo à la place de la pointe chauffante. De plus, pour obtenir les résultats en format tableur exploitable, il faut scanner la feuille imprimée, délimiter les axes et les valeurs. Un programme analyse l'image et en extrait les données. Cela prend beaucoup de temps et peut introduire des imprécisions. Ma mission est de trouver comment faire pour que les résultat arrivent directement sur un ordinateur.

Troisièmement, je dois créer un logiciel de conversion de fichier sortant du goniophotomètre [1] en un fichier compatible avec Dialux.
Le goniomètre est un appareil qui mesure l'éclairement lumineux* d'un luminaire en fonction d'un angle horizontal et vertical. Voir figure 2, ci dessous.

FIGURE 2 – Schéma d'un goniomètre avec les axes de rotations et un capteur

1. Souvent appelé goniomètre ou gonio

Cela permet de connaitre la répartition de la lumière émise par le luminaire.
Le fichier de sortie n'est pas compatible avec Dialux, un logiciel de simulation d'éclairage. Pour l'instant la conversion se fait à la main. Ma mission est de faire un programme qui fait la conversion automatiquement.

	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7	Semaine 8
BancXY	Decouverte LabVIEW + Materiel	Réalisation V1	Réalisation V2 + Panne PC	Finalisation V2 + Installation nouveau PC				
Goniometre			Réalisation V1					
Lambda 9					Tentative de communication	Connection au PC	Décodage des informations	Test et Finalisation
Rapport De Stage							Debut de rédaction	Milieu de rédaction

FIGURE 3 – Tableau temporel de la réalisations des missions

2 Travail effectué

2.1 Banc XY

2.1.1 Problème

Comme expliqué précédemment, après un changement de contrôleur d'axes pour cause de panne, le programme qui se charge d'envoyer les instructions au contrôleur pour faire la mesure n'est plus compatible. Les mesures sont donc effectuées à la main par l'opérateur qui :

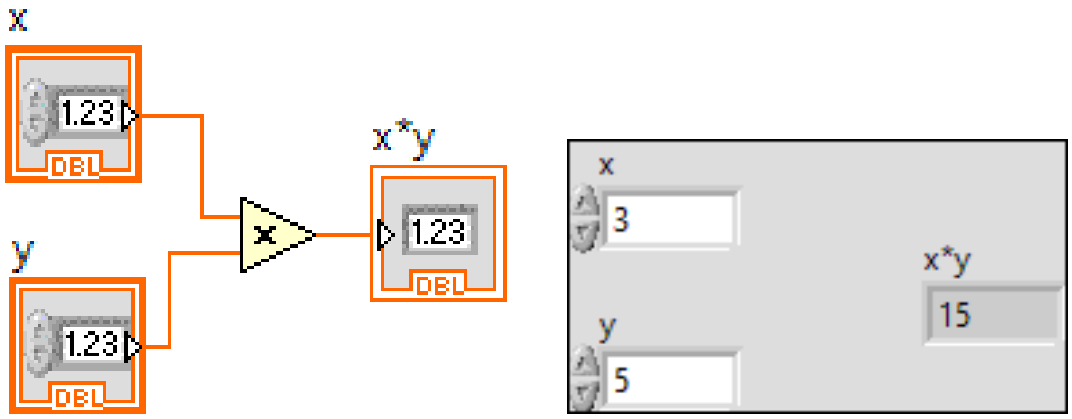
- Affiche la tension de la cellule de mesure en temps réel sur l'ordinateur
- Centre la cellule de mesure sur le centre de la lampe
- Allume la lampe
- Entre les coordonnées dans le contrôleur pour qu'il déplace la cellule
- Retient tension maximale qu'il voit à l'écran
- La note, et recommence pour chaque point du quadrillage ou de la croix.

Cela prend beaucoup de temps, peut introduire des erreurs et incite les personnes demandant des mesures à réduire la taille de leur demande. Le banc XY n'est pas seulement utilisé par le département Qualité pour tester des lampes en sorties de production, mais aussi par le département Développement pour tester des lampes en cours de développement. Les mesures sont souvent plus grande que les mesures de contrôle qualité et ne sont donc plus vraiment effectuées.

2.1.2 Mon travail

L'ancien programme LabVIEW, datant de 2008, possède plusieurs fonctionnalités qui ne sont plus nécessaire. Il me semble plus cohérent de repartir de zéro. Le choix de LabVIEW découle du fait, que c'est le seul langage de programmation que quelqu'un dans l'usine maîtrise, et de ce fait peut maintenir et modifier les programmes.
Ne connaissant pas LabVIEW, j'ai passé les premiers jours à apprendre les bases de ce langage. C'est un langage graphique, qui fonctionne par l'intermédiaire de blocs (VIs). Chaque VI effectue une opération, et les VIs transmettent les données entre eux avec des fils. Ils agissent comme des fonctions.
Les programmes LabVIEW sont constitué d'un diagramme qui représente la partie programmation, et une face avant, qui représente l'interaction avec l'utilisateur, ou

les VIs appelants. Voir figure 4, page 11.
Les entrés d'informations comme `x` et `y` sont appelées commandes, et les sortie comme `x*y` sont appelées indicateurs.



(a) Diagramme (b) Face avant

FIGURE 4 – Exemple de programme LabVIEW

On retrouve dans LabVIEW tous les concepts de base de la programmation, comme les conditions, les boucles, certaines structures de données, les fonctions etc... Il y a même des concepts plus avancés, comme la programmation objet et les event listeners. En revanche il n'y a pas à proprement parler de variable. Elle sont remplacées par des fils qui transmettent les données d'un endroit à l'autre. Cela est dû au parallélisme innée de LabVIEW, qui exécute les morceaux de code non dépendants en même temps. Les fils servent alors à déterminer l'ordre d'exécution, et des variables comme on utiliserait dans un langage séquencé, ne fonctionneraient pas car tout le code s'exécuterait en même temps, sans attendre d'avoir les bonnes valeurs.
Cela, en plus de la programmation graphique qui nécessite de fouiller dans des menus pour trouver les bon VI, m'a pris un certain temps d'adaptation. Malgré tout, je pense que mes 2 années à l'IUT m'ont permis de m'adapter plus rapidement, que si je n'avais pas eu de connaissance en programmation.

Une fois des connaissances de base acquises, je cherche comment communiquer avec le nouveau contrôleur. C'est un NewPort ESP302, qui se branche en série sur l'ordinateur et communique avec le protocole RS232. Il reçoit des mots de commandes

tels que `1PA100` pour déplacer l'axe 1 de 100mm.
En premier lieu je réussis à envoyer des commandes depuis NI MAX, un utilitaire de LabVIEW. Il permet d'établir une communication sans avoir à programmer quoi que ce soit, c'est donc pratique pour vérifier que les paramètres fonctionnent réellement.
Ensuite, j'essaie de faire la même chose depuis un programme LabVIEW. Après m'être rendu compte que les constantes de chaînes de caractères en LabVIEW ne supportent pas par défaut les caractères échappées (`\r`), j'ai réussi à envoyer des commandes au contrôleur.
Entre-temps Alain Collet, la personne chargée de faire des mesures avec le banc, m'a montré une mesure d'irradiance, (*Étapes notées plus haut 2.1.1, page 10*). J'en ai profité pour poser des questions sur le fonctionnement du banc, et sur les besoins pour le nouveau programme. Il m'a montré les résultats de mesure de l'ancien programme, pour que je puisse les reproduire dans le nouveau.

Par la suite, je note les choses basique que doit faire mon programme et que je ne sais pas encore faire :

- Se déplacer en quadrillage ou en croix
- Mesurer une tension de la cellule de mesure et la convertir en irradiance [2]
- Stocker les valeurs dans un fichier tableur

Je commence par faire un programme qui se déplace en zigzag sur un quadrillage de points, ou en croix. Le programme sort une liste d'instructions pour le contrôleur en fonction du mode de déplacement choisi.
A ce moment, je me rend compte que si je veux faire une mesure précise, je doit attendre que les moteurs se soit arrêtés avant de mesurer la tension. Je commence à essayer de trouver une solution pour savoir quand les moteurs sont arrêtés. Le contrôleur ne renvoie pas de message pour dire qu'il a fini de bouger. Je me rabats sur la deuxième meilleur chose, je demande au contrôleur de m'envoyer son statut après que le dernier axe ait fini de bouger. La structure de l'instruction devient donc
`1PA-50;1WS;2PA-40;2WS;TS?\r` où :

- `PA` est une instruction de déplacement de coordonnées absolues de l'axe dont le nombre est situé avant
- `-50` et `-40` sont les coordonnées ou l'axe doit se déplacer

2. La cellule de mesure donne une tension qu'il faut convertir en irradiance avec une formule spécifique à chaque cellule

— `WS` demande d'attendre que l'axe dont le nombre est situé avant ait fini de bouger avant de passer à la commande suivante
— `TS?` demande le statut du contrôleur, le résultat n'est pas significatif car je m'en sert juste comme indicateur de fin de commande

Voici le déroulement de l'instruction :

1. L'ordinateur écrit `1PA-50;1WS;2PA-40;2WS;TS?\r` sur le port série
2. L'ordinateur attend de recevoir une réponse du contrôleur sur le port série
3. Le contrôleur reçoit la commande et commence le déplacement l'axe 1 aux coordonnées -50mm
4. Le contrôleur attend que l'axe 1 ne bouge plus
5. Le contrôleur déplace l'axe 2 aux coordonnées -40mm
6. Le contrôleur attend que l'axe 2 ne bouge plus
7. Le contrôleur renvoie son statut
8. L'ordinateur reçoit le statut et continue le programme

Récupérer la tension de la cellule est simple, car il suffit d'envoyer une instruction `#01\r` sur un port série pour recevoir la tension.

Il ne reste plus qu'à stocker des valeurs dans un fichier tableur. Encore une fois, les fonctions de LabVIEW sont très pratiques, car il existe déjà une fonction qui permet de convertir un tableau en 2 dimensions en un fichier tableur.

J'ai maintenant tous les éléments techniques nécessaires pour commencer le programme entier. Je fais une liste de toutes les fonctionnalités à implémenter et contraintes à respecter :

— Choisir une dimension de mesure 28/02
— Choisir un pas de mesure 28/02
— Il faut que la distance soit un nombre pair pour pouvoir le diviser pour avoir des coordonnées de départ entière 28/02
— Choisir entre mesure d'irradiance en croix ou en cartographie 29/02
— Bouton visualisation des informations de mesure calculées et ajustées (nombre de points, distance de mesure) 29/02
— Il faut que la distance soit multiple du pas correspondant pour obtenir un nombre de point de mesure entier 01/03

- Il doit y avoir un passage par 0, donc il faut que la distance/2 soit multiple du pas correspondant 01/03
- Générer liste d'instruction, et afficher liste de coordonnées avant de lancer la mesure 01/03
- Barre de progression 04/03
- Informations à enregistrer dans le tableur de la mesure :
  - Opérateur 05/03
  - Nom de la source mesurée (nom de la lampe) 05/03
  - Ref de la source mesurée (ref de la lampe) 05/03
  - Tension fournie moyenne 05/03
  - Intensité fournie moyenne 05/03
  - N° SQ (indiqué en haut de la feuille de demande de mesure) 05/03
  - Désignation de la cellule de mesure 05/03
  - Température de la cellule de mesure 05/03
  - Distance lampe cellule 05/03
  - Temps de pause pour mesure 5/03
  - Tension alim ±(commentaires sur ancienne mesures) 05/03
  - Puissance (commentaires sur ancienne mesures) 05/03
  - Intensité (commentaires sur ancienne mesures) 05/03
  - Date de mesure 06/03
- Bouton voir tension cellule de mesure en direct 05/03
- Bouton aller au zéro 05/03
- Bouton mettre à zéro à cette position 06/03
- Boutons déplacer le chariot de 5mm 06/03
- Tableau d'irradiance calculé en fonction de la désignation de la cellule de mesure 06/03
- Gestion des erreurs du boitier ESP302 07/03

Je commence par intégrer la possibilité de choisir les dimensions de la mesure ainsi que le pas de déplacement. C'est-à-dire qu'on peut choisir la distance sur laquelle la mesure sera effectuée sur les deux axes, ainsi que la distance entre chaque points de mesure. Comme on mesure l'irradiance d'une lampe, il est important de passer par le point le plus lumineux de la lampe, souvent au centre de la lampe, au 0. Il faut donc que le programme vérifie que la combinaison de dimensions et de pas de déplacement

permet de passer par le 0. Et d'ajuster si ce n'est pas le cas.
Il faut que la distance soit un nombre pair, pour pouvoir la diviser par 2 et avoir des coordonnées négatives et positives entières. La distance divisée par 2 doit être multiple du pas pour que l'un des points de mesure soit le 0.
J'ajuste donc les distances au supérieur si ce n'est pas le cas.
Exemple :
Distance = 65, pas = 10
On ajuste la distance à 80 car $80/2 = 40$ et 40 est multiple de 10.
Une fois les dimensions ajustées, on les affiches à l'écran pour, que l'utilisateur puisse vérifier que c'est bien ce qu'il veut. J'ai aussi affiché les coordonnées de tous les points de mesure, ainsi que le nombre de points après ajustement.
A ce point, le programme prend des dimensions et des pas, offre le choix de faire une cartographie ou une croix, affiche les coordonnées des points de mesures, et ajuste les dimensions si besoin. On peut lancer le déplacement et la cellule bouge sur toutes les coordonnées.

Pour enregistrer les valeurs dans un tableau excel, il faut d'abord savoir ou les stocker dans le tableau. Avec le déplacement en zigzag et le déplacement en croix, je dois avoir deux méthode de calcul de coordonnées à chaque fois que j'enregistre une valeur.
J'ai résolu ce problème en enregistrant pour chaque instruction de mesure, au moment de sa création, les coordonnées d'enregistrement dans le tableau. A ce moment là du programme je sais exactement ou sera la cellule, il est donc logique de définir ou la stocker dans le tableau.

J'ai les données d'irradiance, il ne me reste plus qu'à ajouter les informations relatives à la mesure, comme le nom de l'opérateur, la date de la mesure, le numéro de mesure spécial, les informations sur la lampe etc… Je fais un menu pour entrer ces informations, et je trouve un moyen de les formater pour les stocker dans le fichier tableur, ce qui n'est pas facile car LabVIEW n'a pas de fonction intégré pour écrire des informations textuelles dans un fichier tableur, je formate donc les informations avec des spécificateurs de format* et je les écris dans un fichier texte. Cela apporte l'avantage d'être facile à concevoir, mais l'inconvénient de ne pas produire un diagramme très lisible. Voir figure 5, page 16.

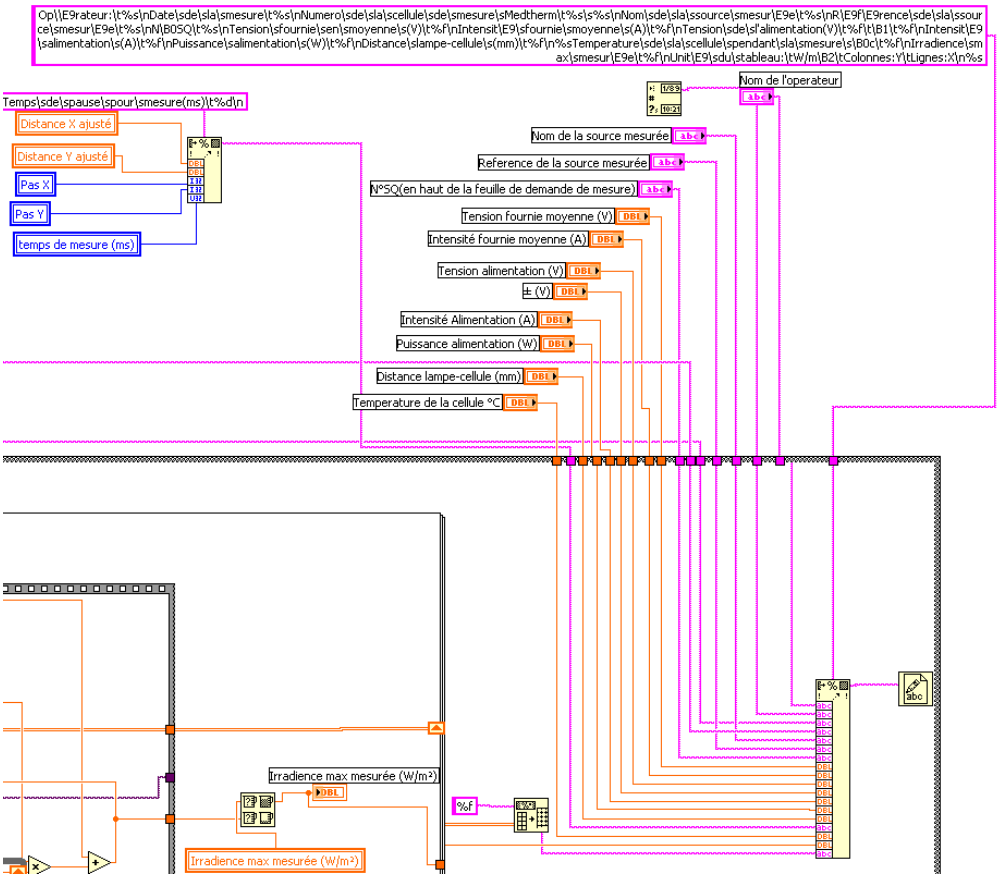


FIGURE 5 – Diagramme contenant les spécificateurs de format

Le seul élément manquant avant d'avoir une première version fonctionnelle est de pouvoir choisir la sonde de mesure utilisé pour que la conversion de la tension en irradiance soit correcte.
Pour cela j'ai choisi d'utiliser un fichier de configuration qui contient les informations des valeurs de conversion pour chaque cellule. Associé à un menu déroulant pour choisir la cellule, le logiciel peut maintenant convertir les tensions en irradiance.

La version est très rudimentaire (l'interface graphique n'est pas pratique à utiliser, aucune facilitation de l'expérience utilisateur) du programme est fonctionnelle. On peut choisir les dimensions, les pas, le type de mesure, rentrer les informations de la mesure, lancer la mesure, et enregistrer les données dans un fichier tableur.
Avec les fonctionnalités de bases implémentées, je dresse une liste des choses à améliorer et à rajouter :

— Améliorer l'interface graphique
— Flèches de déplacement de la cellule, de mise à zéro pour le centrage de la cellule par rapport à la lampe
— Limite sur les données entrées par l'utilisateur
— Commentaires dans le code

Je fais des croquis de l'interface graphique et je regarde quelques sites web et logiciels pour m'inspirer. Je finis par regrouper certaines fonctionnalités ensemble en fonction de leur type, et je les place dans des onglets pour les délimiter.

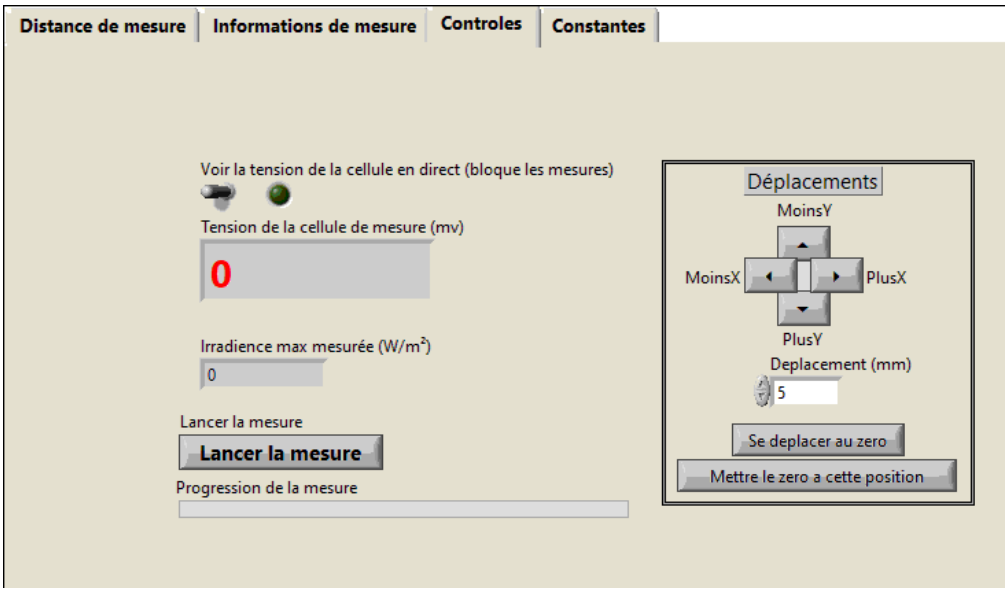


FIGURE 6 – Image de l'interface graphique

Le premier onglet sert à choisir les dimensions, les pas de déplacement et le type de mesure (cartographie ou croix) ainsi qu'a visualiser les coordonnées de mesure et le nombre de points.
Le deuxième onglet sert à entrer les informations de la mesure qui n'influent pas sur le déroulement de celle ci à l'exception du temps de pause pour mesure (temps pendant lequel la cellule reste immobile et la tension est mesurée en continu pour obtenir la valeur maximale).
Le troisième onglet est l'onglet des contrôles, (voir figure 6, page 17,) il permet de voir la tension de la cellule en direct, de déplacer la cellule et de lancer la mesure. J'ajoute des boutons pour déplacer la cellule, et pour mettre le zéro à l'endroit ou la cellule est

actuellement. Cela permet de pouvoir bouger la cellule sans avoir à utiliser l'interface peu pratique du contrôleur.
J'ajoute des limites sur les données entrées par l'utilisateur qui influent sur la mesure, pour éviter les divisions par 0, les nombres négatifs pour des valeurs qui doivent être positives etc… Cela ne se fait pas dans le code mais dans les propriétés des commandes sur la face avant.
Je commence aussi à mettre des commentaires dans le code, qui jusqu'à présent ne servait que de test pour les différentes fonctionnalités.

Le programme est maintenant dans une première version utilisable. Charles RAMMAERT, organise une démonstration du programme en faisant une mesure sur une lampe avec tout le personnel concernés par le banc XY. Après la mesure, les personnes présentes ont fait des retours sur le programme, et j'ai noté les points à améliorer :

- Pouvoir déplacer la cellule et afficher la tension en même temps
- Formatter les données de mesures d'irradiance en croix différemment dans le tableur
- Afficher le graphe d'irradiance en temps réel
- Réduire la taille du diagramme des VIs

Tous ces points sont intrinsèquement liés à la conception du programme.
LabVIEW est un langage graphique très loin des langage plus classique comme Java, Python ou PHP avec lesquels je suis familier. Comme dit précédemment, il n'y a pas de variables, les données transitent par des fils qui sont tirés d'un VI à un autre. Cela rend le programme très illisible car on doit remonter la source du fil pour savoir d'où vient l'information.
Si le VI dépasse la taille de l'écran il faut alors se déplacer sur le diagramme ce qui rend le débogage et la modification assez compliqué. Comme les données commencent toutes dans des commandes, et que ces commandes sont situées sur une seul face avant, elles sont toutes présente dans le diagramme. Par exemple les données à enregistrer avec la mesure sont toutes présentes sur le diagramme, (voir figure 5, page 16), et comme il y en à beaucoup, ce n'est pas très lisible.
La présence d'autant d'informations rend le diagramme indéchiffrable, (voir figure 7, page 19), car on doit tirer un fil de chaque commande jusqu'à chaque VI qui va la traiter. Il faut trouver un autre moyen de faire transiter les données. Il faut aussi trouver un autre moyen de gérer les actions des boutons.

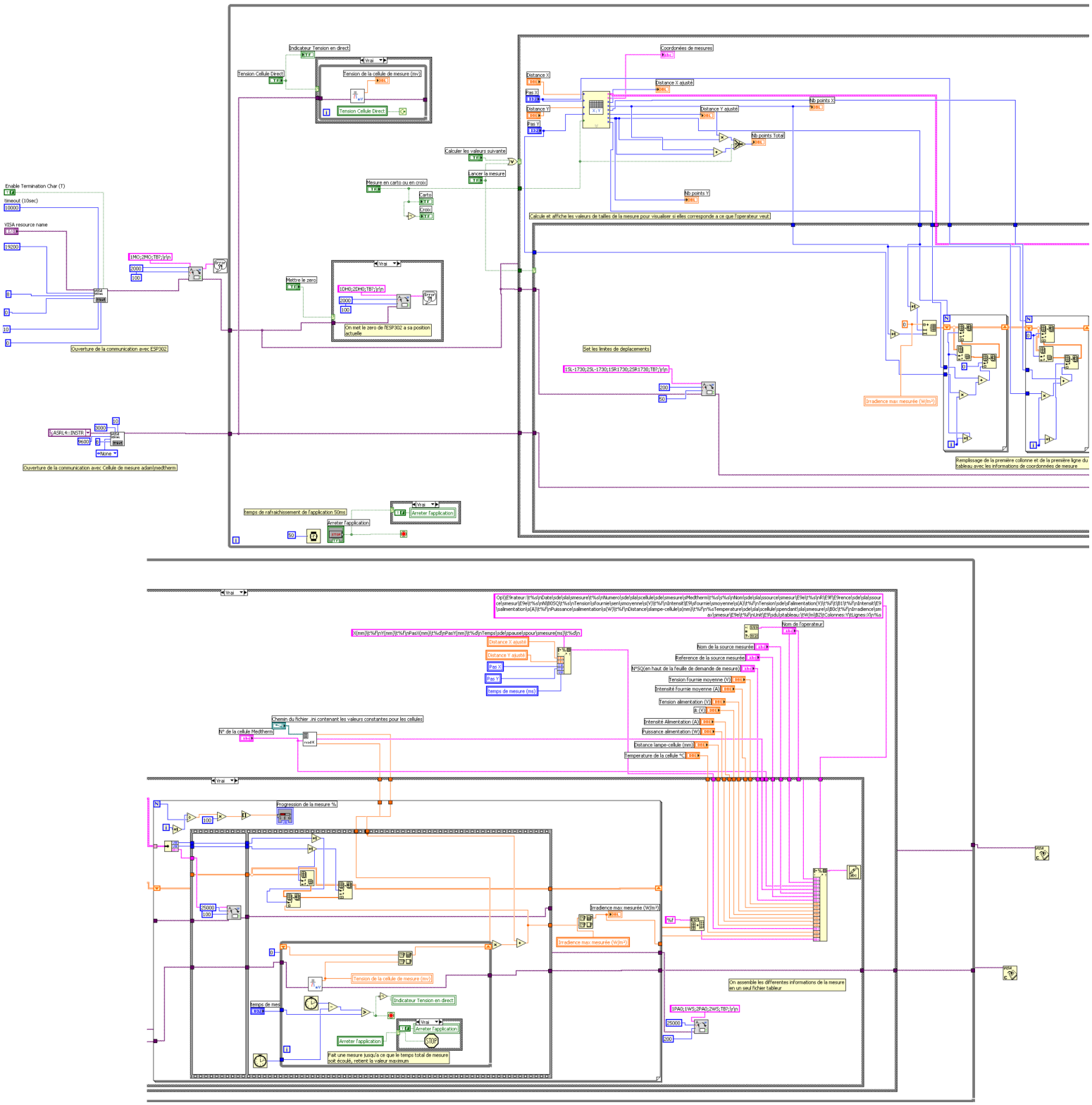


FIGURE 7 – Image diagramme V1 illisible

Pour l'instant les actions sont gérés par des conditions, dans une boucle infinie. Quand un bouton est préssé, pendant qu'il exécute du code, le reste de l'application est bloqué. Ma première idée pour gérer les actions, c'est les "Event listeners", il existe quelque chose de similaire dans LabVIEW, les structures d'évènements. Elles permettent de définir l'exécution de code en fonction d'évènements configurés. Cela permet d'avoir plusieurs boutons cliquables en même temps, ce qui résout le problème d'afficher la tension et de pouvoir bouger en même temps.

Le deuxième problème est le nombre de fils et leur longueur qui rende le tout illisible. Pour diminuer le nombre de fils allant de VI à VI je decide d'utiliser des clusters. Ce sont des structures de données comme des objets mais on n'y stocke pas de fonctions ou de méthode. Ces clusters regroupent les entrées de données en un seul fil qu'on peut grouper et dégrouper.
Cela permet de réduire le nombre de fil se baladant d'un VI à l'autre mais pas à l'intérieur meme d'un VI. En effet si on veut traiter les données, il faut inévitablement tirer un fil qui va du dégroupement du cluster jusqu'au VI traitant les données. Cela sert donc simplement à réduire la taille des VI, mais pas vraiment leur complexité. (Voir figure 8, page 21). Le fait de ne plus avoir à trop se déplacer pour voir d'où vienne les informations constitue déjà une grosse amélioration par rapport au VI qui faisait 5 écrans de large.

Ces avancées, combinées au fait de diviser le programme en plus de sous-VIs font que la lisibilité a grandement augmenté. Une fois ces changements effectués, la modification du programme est devenus plus simple, ce qui m'a permis de rapidement modifier le formatage des données des mesures en croix dans le tableur, et d'ajouter un graphe d'irradiance en direct lors de la mesure, qui permet de verifier visuellement que la mesure se déroule comme prévue (voir si il y a une chute inattendue d'irradiance lors de la mesure).

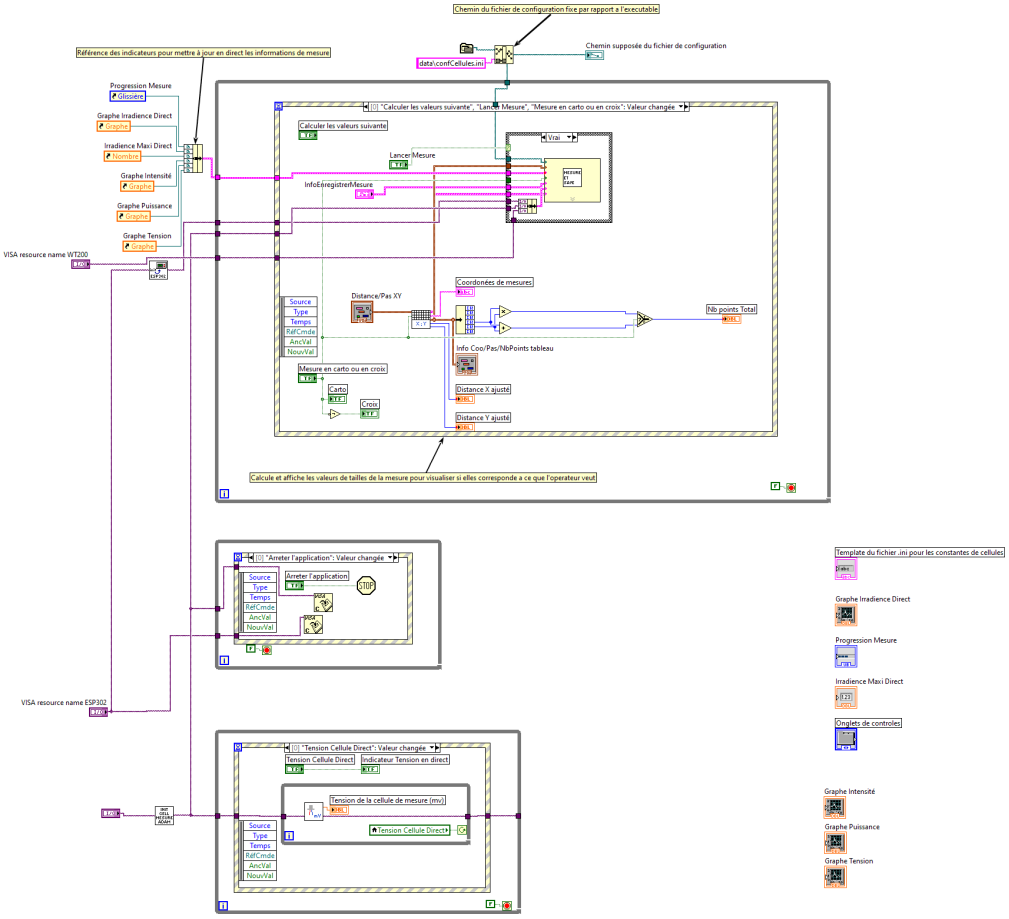


FIGURE 8 – Exemple du diagramme du même VI que celui de la figure 7, en utilisant des cluster et en augmentant le nombre de sous-VIs

Les problèmes réglés, on a fait un autre test qui a révélé qu'il fallait aussi intégrer la courbe du multimètre en direct pour surveiller les potentiels chutes de tension faussant la mesure. Et après ça seulement un seul bug à été rapporté, il s'agissait du VI qui attendait d'avoir le status du contrôleur pour passer à la suite qui ne fonctionnait pas toujours.

Les étapes de déroulement étaient, on envoie une instruction au programme et quand on détecte que le buffer de sortie n'est pas vide, on s'arrête. (Voir algorithme 1, page 22.) Le problème était que le programme s'arrêtait juste après avoir envoyé l'instruction, même si l'instruction mettait 5 secondes au contrôleur à completer. Le plus bizarre étant que le problème ne survenait plus si on mettait un temps d'attente entre la

détection du buffer non vide et l'arrêt, alors qu'une fois le buffer non vide detecté, 35 ms ne devrait pas se remarquer. (Voir algorithme 2, page 22), La seule différence avec l'autre est l'instruction soulignée.
Normalement une fois que le buffer non vide est détecté le programme devrait s'arrêter immédiatement, la seule différence et 35ms d'attente après avoir détecté que le buffer n'est pas vide. Je n'ai pas réussi à vraiment comprendre de ce comportement, j'ai réussi à le contourner de manière fiable. Je blâme LabVIEW et son parralélisme abusif.

Algorithm 1 Programme s'arrétant immédiatement après l'envoie de l'instruction

```
EnvoyerInstruction(instruction)
instructionFini ← false
while instructionFini is false do
    if BufferEtreVide() is false then
        resultatBuffer ← LireBuffer()
        instructionFini ← true
    end if
end while
```

Algorithm 2 Programme s'arrétant à la fin de l'éxecution de l'instruction par le contrôleur

```
EnvoyerInstruction(instruction)
instructionFini ← false
while instructionFini is false do
    if BufferEtreVide() is false then
        resultatBuffer ← LireBuffer()
        Attendre(35ms)
        instructionFini ← true
    end if
end while
```

Après ce bug corrigé, le programme est considéré comme fini. J'ai rédigé la documentation utilisateur pour étaler clairement le déroulement des étapes et quoi faire en cas de problème. J'ai écris la documentation technique qui décrit le système et ces particularités.
J'ai finalement terminé d'écrire les commentaires du programme.
Le banc XY est fonctionnel. Le gains de temps est considérable, les mesures prennent environ 4 fois moins de temps que lors ce qu'elles étaient faites à la main.

2.2 Conversion des fichiers du goniomètre

2.2.1 Problème

Le goniomètre, est un appareil qui permet de mesurer l'éclairement lumineux d'un luminaire, à différents angles horizontaux et verticaux, pour en faire une cartographie. Cela permet ensuite de visualiser la zone d'éclairement du luminaire dans un logiciel spécialisé, Dialux.
Le problème est que pour l'instant le logiciel contrôlant le goniomètre sort un fichier tableur très simple avec les valeurs pour chaque coordonnées d'angles, et ce fichier n'est pas compatible avec Dialux.
Dialux prend un fichier `.ies` respectant la norme `ANSI/IESNA LM-63-02`. Pour l'instant la conversion est faite manuellement, ce qui prend beaucoup de temps et est très minutieux.
Ma tâche est de faire un programme qui convertisse le fichier tableur en fichier `.ies` respectant la norme `ANSI/IESNA LM-63-02`.

2.2.2 Mon travail

Je commence par lire la norme `ANSI/IESNA LM-63-02` pour comprendre ce que je dois faire. Le document est long mais très détaillé, ce qui me facilitera la tâche car il ne laisse aucune place au doute. Je note les points importants pour la conversion :

- L'en-tête du fichier doit contenir des informations sur le luminaire, la date de la mesure, le fabriquant etc…
- Ensuite il y a des valeurs séparé par des virgules qui corresponde au nombre d'angles verticaux et horizontaux, informations diverses sur la mesure et le goniomètre etc…
- Enfin il y a les valeurs d'éclairement d'angle verticaux pour un angle horizontal par ligne, converti en intensité lumineuse*
- Toutes les combinaisons d'angles de départ et d'arrivée ne sont pas possibles

La manière dont est formatés les angles dans le fichier `.ies` est assez différente du fichier d'origine, selon les angles de départ et d'arrivés choisis sur la mesure il faut manipuler le tableau différemment. Les valeurs d'angles de départ et d'arrivées autorisés par la norme pour les angles verticaux et horizontaux sont 0 ;90 et 90 ;90.
Cela laisse 6 combinaisons d'angles de départ et d'arrivées possibles, ainsi que les

opérations qu'il faut faire sur le tableau des valeurs et des angles, sur les deux axes, pour qu'il respecte la norme :

- 0 ;90 On ne fait rien
- 0 ;-90 On convertit les valeurs d'angles en valeur absolue
- 90 ;0 On renverse le tableau, angles et valeurs comprises
- 90 ;-90 On renverse le tableau, angles et valeurs comprises
- -90 ;0 On renverse le tableau, angles et valeurs comprises et on convertit les valeurs d'angles en valeur absolue
- -90 ;90 On ne fait rien

Avec ça j'ai tout les éléments nécessaire pour commencer le programme. Le programme est aussi fait avec LabVIEW, toujours pour la même raison, c'est le seul langage de programmation que quelqu'un dans l'usine maîtrise. Le programme et en soit assez simple une fois qu'on à compris comment manipuler les angles et les valeurs. Le reste est du formatage d'entrée d'utilisateur en informations conforme à la norme. J'utilise mes connaissance sur la manipulation de texte et la construction de fichier tableur acquises lors de la réalisation du programme du banc XY.
Tout au long du développement j'utilise un fichier de mesure avant et après conversion validé auparavant avec Dialux, pour vérifier que le programme fonctionne bien. Une fois terminé, j'envoie le fichier converti avec mon programme à quelqu'un qui peut vérifier sa validité avec le logiciel Dialux. Ce travail plutôt cours ne m'a pris que 3 jours.

2.3 Spectrophotomètre Lambda 9

2.3.1 Problème

Comme expliqué dans l'introduction, le Lambda 9 imprime ses résultats sur une imprimante propriétaire. Les résultats doivent ensuite être scannée manuellement puis manipulés à l'intérieur d'un logiciel pour enfin avoir les valeurs en tableur. Si jamais l'imprimante tombe en panne, le Lambda 9 devient inutile. Ma mission est de trouver un moyen d'enlever l'imprimante et de récupérer les données directement sur un ordinateur.

2.3.2 Mon travail

L'idée de remplacer l'imprimante thermique par un ordinateur, qui émulerait l'imprimante et enregistrerait les données dans un fichier tableur. Pour ce faire Florian Jean Pierre, l'initiateur du projet, a acheté un convertisseur parallèle/USB pour connecter l'ordinateur au Lambda 9.
Cependant il c'est vite avéré que ce n'était pas si simple que ça. Les ports parallèles ne sont pas faits pour émuler une imprimante. Quand je me rends compte de cela je commence à chercher des solutions pour émuler une imprimante avec un port parallèle. Je trouve rapidement quelques produits qui permettent de faire ça, mais en raison de l'âge du Lambda 9 et du fait que l'imprimante soit spécifique, je doute fortement de la compatibilité de ces produits.
C'est à ce moment que Olivier Rouyer, un membre du service Qualité, me donne un HP 4951C Protocol Analyzer. (Voir figure 9, page 26.) C'est un appareil qui permet d'intercepter les données qui passent entre deux appareils, et de les afficher, ainsi que l'état de certain signaux.

FIGURE 9 – HP 4951C Protocol Analyzer

FIGURE 10 – Lambda 9 et Protocol Analyzer

Malgré le fait que l'imprimante soit connectée au Lambda 9 avec un câble DB25 [3], le même port utilisé habituellement pour les imprimantes, ce n'était pas un protocol de transmission de données en parallèle qui était utilisé, mais le protocole série RS232. Je m'en suis rendu compte après avoir fait fonctionné le protocol analyzer et avoir vu

3. Ports D-subminiature utilisés pour la communication parallèle et série

qu'il est marqué RS232 sur le boitier.
Une fois le protocol analyzer branché sur la connexion Lambda 9 → imprimante avec un cable Y, (voir figure 10, page 26), il a pu faire une détection automatique des paramètres de communications, et afficher les données qui passaient entre les deux appareils.

Je vois les données, mais elles sont sur le protocol analyzer et l'imprimante doit toujours être branchée pour les voir. Dans le manuel du protocol analyzer il est marqué qu'il peut simuler un DCE ou un DTE. Cela correspond exactement à ce qu'on veut faire, simuler une imprimante.
Je réussi d'abord à simuler le Lambda 9 avec l'imprimante, et je réussi à la faire imprimer des données. J'ai mis plus de temps à comprendre comment simuler l'imprimante avec le Lambda 9, car je n'avais pas compris que le Lambda 9 attendait une réponse à la fin de chaque chaine de caractères. J'ai observé la communication entre le Lambda 9 et l'imprimante, et des trames de données envoyées de l'imprimante au Lambda 9 et inversement, j'ai déduit les bases du protocole et j'ai configuré le protocol analyzer pour simuler l'imprimante avec ce programme :

```
1  Simulate DCE
2
3  // on allume les signaux DSR CD et CTS pour dire que l'
       imprimante est prête
4  Block 1
5  Set Lead DSR On
6  and then
7  Set Lead CD On
8  and then
9  Set Lead CTS On
10 and then
11 Send F,00\r  // instruction qu'envoie l'imprimante
12
13 Block 2
14 When DTE \r  // quand le Lambda 9 envoi un retour chariot
15 then Goto Block 3  // on va à l'étape suivante
16
17 Block 3
18 Send 01\r  // on envoie l'instruction qui dit que ça c'est bien
       passé
19 and then
20 Goto Block 2  // on revient à l'étape précédente pour attendre
       un retour chariot
```

Ce programme simule l'imprimante qui au démarrage envoie la chaine `F,00\r`, et qui répond `01\r` quand le Lambda 9 lui envoie un retour chariot. Le programme fonctionne, quand on débranche l'imprimante, le Lambda 9 envoie les données au protocol analyzer. J'ai réussi à confirmer que l'émulation de l'imprimante et la capture des données étaient possible. Maintenant il faut le faire sur un ordinateur pour récupérer les données.
Pour ça j'ai installé un ordinateur et j'ai commencé à essayer de le faire communiquer avec le Lambda 9. Malgré un programme qui réplique exactement ce que fait le protocol analyzer quand il simule l'imprimante, je n'arrive pas à faire fonctionner la communication.
Après avoir lu les dessins techniques du port imprimante présent dans le manuel du Lambda 9, je me questionne sur l'adaptateur DB25 → DB9 que j'utilise pour connecter l'ordinateur au Lambda 9.
En regardant le câblage de l'adaptateur je me rend compte que les fils de données ne sont pas croisés, le Lambda 9 envoie les données sur le même fil où l'ordinateur les envoie, idem avec les données reçues. Il faut utiliser un cable null modem [4] pour connecter l'ordinateur au Lambda 9. La raison est que ce sont deux DTE qui communiquent ensemble, il faut croiser les fils de données pour que l'envoi de données de l'un aille dans la réception de données de l'autre.
Mais à ce moment je ne dispose pas de câble null modem, et même si j'en avais un, il y a des câblages différents pour les signaux autre que les données qui n'auraient pas forcément fonctionné en raison des capacités de l'ordinateur à générer certain signaux sur certains pins. Je trouve le câblage fonctionnel à terme de nombreuses expérimentations avec des adaptateurs, des câbles et des câblages différents, encore une fois grâce au protocol analyzer qui permet d'utiliser des jumpers pour créer son propre câblage. Voir figure 11, page 29. A ce moment là je ne comprend pas encore toute les subtilités des différents signaux et de la communication, c'est pour ça que je pense nécessaire d'avoir un câblage sur mesure. Je me rend compte plus tard qu'il existe des cables null modem existant qui fonctionneraient probablement très bien. Ce câblage est constitué d'un cable null modem classique qui croise les fils de données, et de 3 jumpers qui permettent d'alimenter les signaux DSR, CD et CTS du Lambda 9. Sans ces signaux le Lambda 9 ne détecte pas la présence d'une imprimante et ne communique pas.

4. Câble qui croisent le fils de données, et certains fils de signaux pour que deux appareils utilisant les même connexions sur le port série puisse communiquer

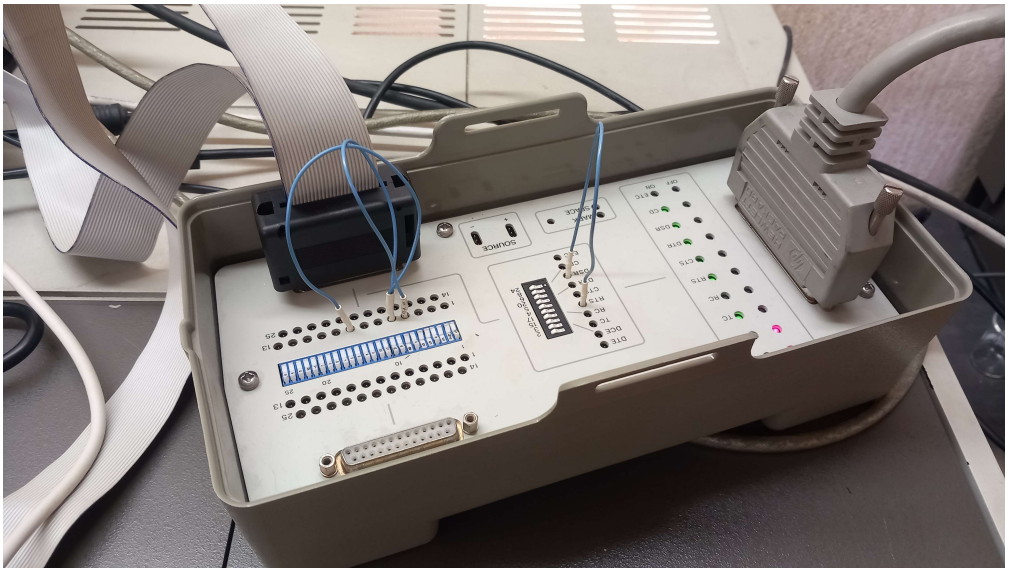


FIGURE 11 – Câblage null modem fonctionnel sur le protocol analyzer

Ce câblage est fonctionnel mais il est constitué de plusieurs convertisseurs et des parties qui ne doivent pas rester là sur le long terme. Je soude un câble ayant le même câblage pour pouvoir remplacer le système temporaire. Voir figure 12, page 29 et figure 13, page 30.

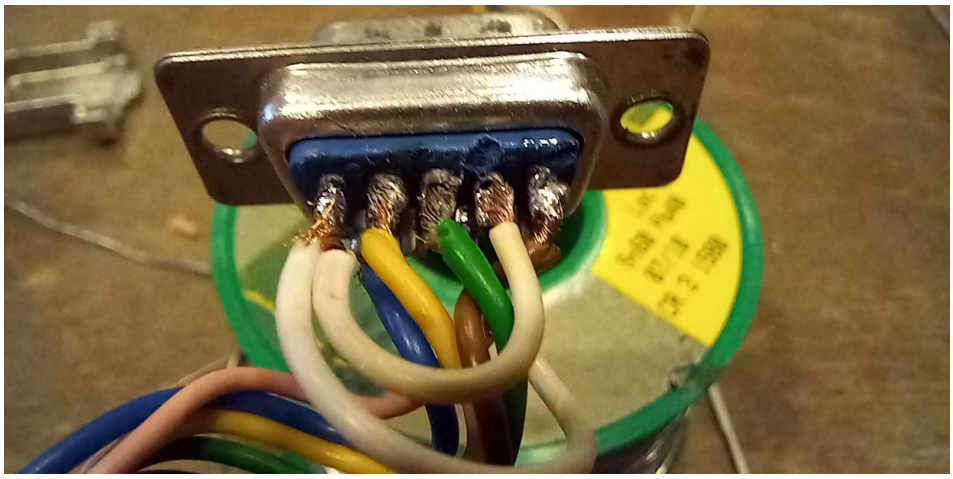


FIGURE 12 – Connecteur soudée

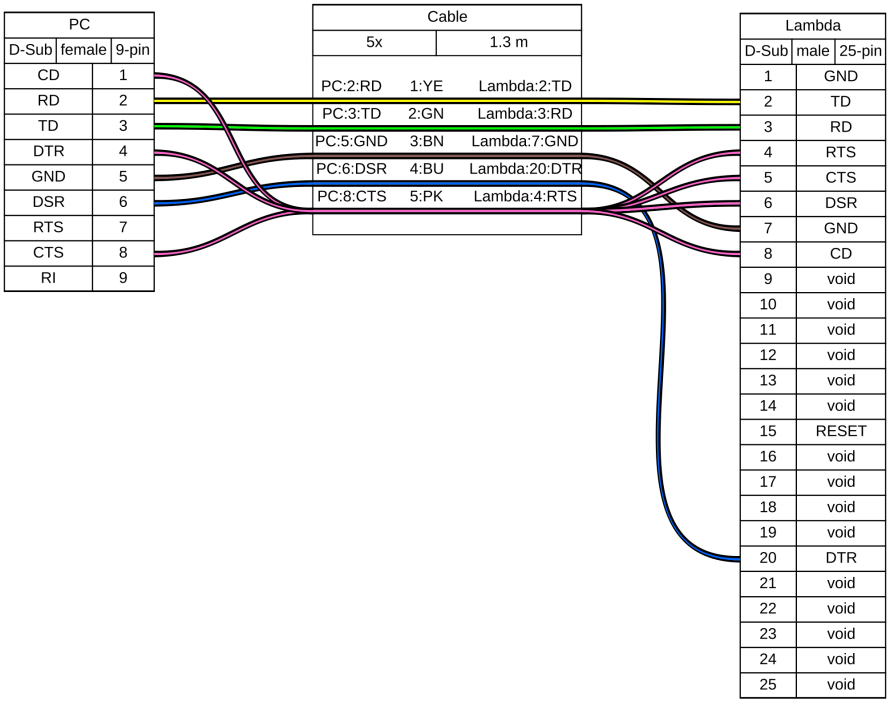


FIGURE 13 – Schéma de câblage

J'ai maintenant un ordinateur qui reçoit les données du Lambda 9. Avant de commencer à écrire un programme convertit les données en tableur, je dois analyser les données reçues pour comprendre comment elles sont formatées.
Ce sont des chaines de caractères ASCII qui disent à l'imprimante quoi imprimer et comment. Ces chaines ne correspondent pas à un langage d'imprimante standard, c'est spécifique au Lambda 9. En voici un court extrait :

```
1 Z0
2 IT,Z0,F15936,416,0,200,D0128,1280,A1,X2100,-100,5,S2090.0,D1,1,
      Y110.0,-22.000,4,Z0,D0128,1280,L1
3 14370
4 14357
5 T,D-1,1,Y110.0,-22.000,4,A0,M0,50,V-2
6 A0,T,V-2
```

Mon but est de pouvoir déduire tous les éléments essentiels de la mesure, comme la longueur d'onde maximum et minimum, l'échelle maximum et minimum, la vitesse de scan les valeurs mesurées. Je procède méthodiquement en faisant de nombreuses mesures en ne changeant qu'un seul paramètre sur le Lambda 9 à chaque fois pour

voir qu'est ce qui change dans les informations transmises à l'imprimante.
Après plus de 100 mesures, je parviens à déduire les informations nécessaires des données transmises. Je rédige un document détaillant quels paramètres du Lambda 9 changent quels valeurs dans les données transmises et quels sont les formules pour les convertir.
Voir figure 14, ci dessous.

```
1 Z0
2 IT,Z0,F15936,416,0,200,D0128,1280,A1,X2100,-100,5,S2090.0,D1,1,
    Y110.0,-22.000,4,Z0,D0128,1280,L1
```

Les valeurs sont séparés par des virgules et on la signification suivante[5]:

Z0

IT

Z0

F15936 *ValEchelleMax*, la valeur réel qui correspond à l'échelle maximal des données envoyées, ici 15936.

416 *ValEchelleMin*, la valeur réel qui correspond à l'échelle minimal des données envoyées, ici 416.

0

200

D0128 *FacteurVitesse*, facteur de vitesse de balayage , ici 128. Change quand la vitesse de balayage (`SCAN SPEED`) ou le format de l'abscisse (`ABSCISSA FORMAT`) change. **IMPORTANT**, pour certaines valeurs de `ABSCISSA FORMAT`, le facteur de vitesse ne change pas avec `SCAN SPEED`, exemples en Table 2. Il est impossible de déduire avec certitude la vitesse de balayage pour certaines valeurs de `ABSCISSA FORMAT`.

1280 *InconnuAbscisse*, valeur inconnue qui change quand `ABSCISSA FORMAT` change, voir Table 2, ici 1280.

A1

[5]Les valeurs qui ne sont pas expliquées sont des valeurs qui ne change pas peu importe les paramètres de mesure

3

FIGURE 14 – Extrait du rapport sur le Lambda 9

Après cela je crée un programme LabVIEW qui analyse les données, fait les conversion nécessaires et les formate en tableur. A ce moment je commence à bien comprendre LabVIEW, j'ai déjà fait le programme du bancXY et celui du Goniomètre. Le développement est assez rapide et sans surprise.
Après une demo avec les utilisateurs finaux, et le développement des modifications demandées, un graphe qui rapporte la mesure en directe ainsi que des virgules à la place des points dans le tableur pour une meilleure compatibilité avec excel, le projet est terminé.
Ce projet constitue un gain de temps assez important car la personne lançant la mesure n'a plus besoin de surveiller l'impression des résultats, ni de faire des manipulations avec la feuille de résultat.

2.4 Panne ordinateur

Au début de la 3ème semaine, après avoir fait la demo du banc XY, l'ordinateur du banc tombe en panne. Il s'arrête de fonctionner au bout de quelques instants après avoir démarré. J'essaye de nombreuses solutions pour le réparer, comme changer les composants, les cables, etc. Je finis par diagnostiquer une carte mère défectueuse. Malheureusement à cause de la licence de LabVIEW qui est liée à la carte mère, il est impossible de changer la carte mère en gardant la licence active.
Cela tombe la semaine ou aucun des deux membres du département Informatique n'est là. Il est impossible de changer l'ordinateur ou de rectifier la licence LabVIEW. Je passe la semaine à travailler sur le programme du banc XY sur un autre ordinateur, mais sans pouvoir le tester. La réinstallation de LabVIEW sur un ordinateur plus récent la semaine d'après n'est pas simple. Il faut réinstaller Windows pour outrepasser l'antivirus qui bloque le lecteur CD.
Une fois l'ordinateur réparé, je passe une journée entière à essayer de faire fonctionner la communication avec la cellule de mesure. Il s'avère que certains paramètres de communication série par défaut avait changé entre LabVIEW 8 et LabVIEW 15, ces paramètres sont cachés, et ne peuvent plus se changer depuis l'utilitaire NI MAX, que j'utilise pour tester le fonctionnement des appareils sans avoir à faire un programme. J'ai du trouver un moyen de comparer les anciennes valeurs de configuration avec les nouvelles pour les remettre à l'identique dans le programme.

3 Conclusion

Lors de ce stage, j'ai créé un programme qui contrôle un outil de mesure pour effectuer des cartographies d'irradiance. J'ai également remplacé l'imprimante d'un spectrophotomètre par un ordinateur. Grâce à cette expérience, j'ai découvert le monde de l'industrie ainsi que les besoins d'une telle entreprise.
Ce stage était à l'origine destiné à un étudiant en BUT Mesure Physique. Cependant, le sujet initial du stage, le banc XY, relevait davantage de l'informatique que de la mesure physique. En effet, il s'agissait d'un problème logiciel pour lequel des connaissances en mesure physique n'étaient pas nécessaires. Je pense même avoir été plus efficace qu'un étudiant en mesure physique n'aurait pu l'être, car mes connaissances avancées en programmation/conception étaient essentielles pour résoudre ce problème.
De plus, les deux autres projets, le logiciel de conversion de fichier du goniomètre et le remplacement de l'imprimante du Lambda 9, étaient purement informatiques.
Particulièrement, le projet du Lambda 9 était assez urgent car la méthode précédente d'acquisition des données via l'imprimante puis un scan devenait de moins en moins fiable, cet outil étant essentiel pour le Développement. Je doute qu'un étudiant en mesure physique aurait réussi à accomplir ce que j'ai fait pour ce projet. Il nécessitait des connaissances en informatique ainsi qu'un certain état d'esprit pour résoudre les problèmes et progresser.
Grâce à ce stage, j'ai découvert LabVIEW, un langage de programmation graphique très différent des autres langages que je connaissais, ce qui a changé ma façon d'aborder certains problèmes. J'ai également perfectionné mes compétences en recherche sur internet pour trouver des documentations difficiles d'accès ainsi que des informations très spécifiques.
En conclusion, j'ai beaucoup apprécié ce stage. Bien qu'il diffère d'un stage en informatique classique, j'ai beaucoup appris et j'ai pu mettre en pratique mes compétences en programmation. Il semble que l'entreprise soit plutôt satisfaite de mon travail, les projets que j'ai réalisés ont entraîné des gains de temps conséquents pour les utilisateurs finaux.

Glossaire

intensité lumineuse L'intensité lumineuse est l'éclairement d'une source lumineuse dans une direction donnée, mesurée en Candela (cd) . 23

irradiance L'irradiance est une puissance de rayonnement éléctromagnétique par unité de surface (W/m^2), la plupart du temps on l'utilise pour décrire la quantité d'énérgie reçue. Par exemple, sur terre, à midi, le soleil à une irradiance d'à peu près $1000W/m^2$. 7

spécificateur de format Un spécificateur de format est un caractère qui commence par un signe de pourcentage (%) et qui est suivi par un ou plusieurs caractères qui définissent le format d'une valeur lors de sa conversion en texte ou de texte en valeur. Par exemple la valeur 143 affiché avec le spécificateur de format `%x` affichera 8F, car 143 en hexadécimal est 8F.
C'est utile aussi pour choisir l'affichage des nombres à virgules . 15

éclairement lumineux L'éclairement lumineux est la quantité de lumière reçue par unité de surface, mesurée en Lux (lx) . 8, 23