

Rapport de fin de 1ère année :

Intégration, pilotage et test d'instruments de mesures photométrique innovants

Talla DIAGNE

Tuteur d'apprentissage : Xavier LEBOEUF
Tuteur académique : Mathieu HEBERT

CEA Leti

Août 2023

Table des matières

1 Mesures IVL de microLEDs sur wafer	2
1.1 Pilotage SMU	2
1.2 Automatisation des mesures par Python avec une interface graphique	2
2 Spectro-imageur Rembrant	3
2.1 Description rapide de l'instrument	3
2.2 Descriptions des fonctions d'analyses automatiques fournies par ELDIM	3
2.3 Exemples d'analyses	4
2.4 Emulateur et pilotage par Python	5
2.5 Développement d'une bibliothèque pour piloter le déplacement d'une platine XYZ motorisée	6
3 Traitement d'images	7
3.1 Recensement de méthodes de caractérisation d'écrans	7
3.2 Génération paramétrable d'images	8
3.2.1 Résolution	8
3.2.2 Détection des défauts	8
3.3 Développement d'un programme d'analyse d'image	9
Perspectives	10
A Annexes	11
A.1 Fonctions proposées par l'émulateur fourni par ELDIM	11
A.2 Lien vers les codes Python utilisés	11

Table des figures

1 Illustration d'une transition radiative	2
2 Pose de pointes sur un wafer	2
3 Mesures de caractérisations sur le Banc Rembrant	3
4 Traitement d'une image réelle par l'émulateur	4
5 Évolution de la détection des défauts en fonction de certains paramètres	5
6 Interface graphique pour le pilotage d'une platine motorisée	6
7 Image d'une mire à 1 pl/mm	8
8 Image fictive de pixels aléatoirement brillants	8
9 Traitement d'une image fictive par un émulateur	9
10 Traitement d'une image (pixels noirs et blancs) par un émulateur	9
11 Traitement de l'image Green avec Python	10

1 Mesures IVL de microLEDs sur wafer

Les µLEDs (Light-Emitting Diode) sont avant tout des jonctions P-N, c'est-à-dire des juxtapositions de deux semi-conducteurs dopés de type P-insertion d'atomes accepteurs- et N-insertion d'atomes donneurs-, capables d'émettre des photons . En polarisant en sens direct cette jonction, on excite des électrons de la bande de valence vers celle de conduction. Lors de leur désexcitation, ils perdent leur énergie sous forme de photons : il s'agit d'une émission radiative. En contrôlant donc la quantité de courant apportée, une µLED peut émettre plus ou moins de photons.

1.1 Pilotage SMU

Le *Source Measure Unit* (SMU) est un appareil qui a la capacité de générer simultanément une tension ou un courant tout en effectuant des mesures précises de ces mêmes paramètres électriques. Cet outil essentiel à la caractérisation de µLEDs est constitué de deux voies -ou channels-. Dans notre cas, la première est reliée à des pointes afin de polariser et mesurer, tandis que la seconde est réservée à la photodiode. Les pointes sont déplacées par des micropositionneurs avec une précision de l'ordre inférieur au micromètre. Cela est nécessaire afin de venir les positionner sur les pads (zones de contact métallique du µLED représentant l'anode et la cathode) et ainsi établir une connexion électrique [2]. En faisant varier la tension appliquée, on mesure l'intensité du courant ainsi que la luminance. On obtient alors deux courbes $I = f(V)$ et $L = f(V)$: on parle de courbe IVL quand on les superpose.

Plusieurs paramètres interviennent lors de ces mesures, et il est important de les connaître afin d'optimiser les résultats. On a par exemple :

- La compliance : valeur maximale de courant pour laquelle la µLED peut fonctionner sans risque de dommage. Aucun courant n'est donc délivré au-delà de la compliance
- Le temps d'intégration : durée pendant laquelle les mesures sont intégrées. Un temps d'intégration trop court peut conduire à des mesures bruitées et peu fiables, tandis qu'un temps trop long peut ralentir le processus de mesure
- Le délai : période entre les mesures successives. Il est nécessaire qu'il soit suffisamment grand afin de laisser le temps à la µLED d'atteindre un état stable (\sim quelques ms)

1.2 Automatisation des mesures par Python avec une interface graphique

On souhaite acquérir les données brutes (le courant, la tension, le photo-courant) fournies par le SMU afin de les analyser et extraire les principales caractéristiques des LEDs testées. La communication entre l'ordinateur et le SMU se fait via un câble GPIB et les commandes peuvent être directement envoyées en TSP (Test Script Processing, langage propre au SMU) par protocole VISA sous cette forme par exemple :

```
1 smua.source.output = 1
```

Toutes les fonctions que j'ai créées ont été enregistrées dans une librairie nommée *COM_SMU*. J'ai également conçu une interface graphique avec *Python* et *QtDesigner*. L'interface permet un contrôle du générateur soit par le courant, soit par la tension. Les données brutes peuvent être enregistrées puis automatiquement analysées afin de fournir plusieurs informations : le flux Φ en Watt puis en Lumen, l'intensité lumineuse I , la luminance L , l'*EQE* (External Quantum Efficiency), le *WPE*. L'analyse est régie par des paramètres et des constantes : la surface A , le photo-courant i , la tension v , la sensibilité de la photodiode S , l'efficacité lumineuse relative spectrale K , l'angle solide Ω . Les formules à la base de ces conversions sont les suivantes :

$$\Phi(W) = \frac{i(A)}{S(A/W)} = \frac{\Phi(lm)}{K(lm/W)}; \quad I(cd) = \frac{\Phi(lm)}{\Omega}; \quad L(cd/m^{-2}) = \frac{I(cd)}{A(m^2)}; \quad EQE(\%) = \frac{\Phi(W)}{i(A)} \times R; \quad WPE(\%) = \frac{\Phi(W)}{i(A) \cdot v(V)}$$

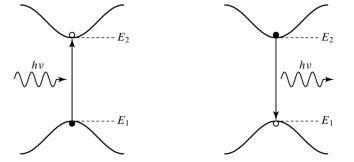


FIGURE 1 – Illustration d'une transition radiative

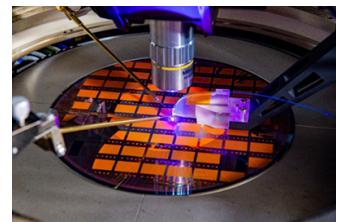
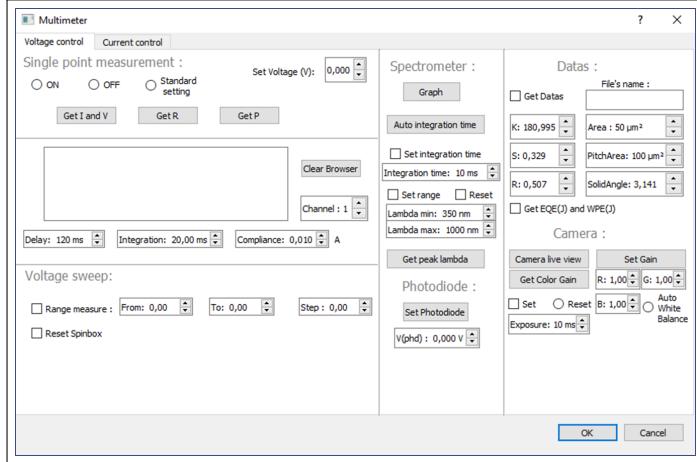
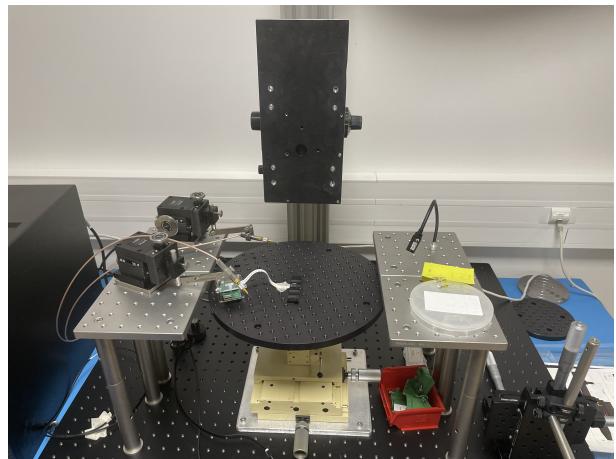


FIGURE 2 – Pose de pointes sur un wafer



(a) Pilotage SMU, spectro et caméra, acquisition et analyse de données, avec interface graphique



(b) Banc Rembrant

FIGURE 3 – Mesures de caractérisations sur le Banc Rembrant

Ci-dessus, une photo de l'interface graphique en question et du Banc Rembrant. C'est sur ce banc que l'on va réceptionner le spectro-imageur et mener les premiers tests afin de vérifier sa conformité. Il est constitué de deux supports horizontaux sur les côtés et d'un troisième qui lui est mobile et vertical.

L'interface permet de faire des mesures en un point unique ou alors un balayage entre deux bornes avec un certain pas entre deux mesures. On remarque la présence des paramètres mentionnés plus haut : le temps d'intégration, le délai, la compliance. A côté du choix de la voie, un tableau de texte permet de communiquer des informations à l'utilisateur. Pour le contrôle du spectromètre QEPro Ocean Optics, on peut choisir encore un fois le temps d'intégration ou les bornes de longueurs d'ondes lors de la mesure. L'acquisition des données se fait quant à elle dans la partie *Datas* : il suffit de cocher la case *GetDatas* puis de donner un nom au fichier texte. Les constantes permettant l'analyse des données brutes peuvent être modifiées, et on peut si on le souhaite obtenir les courbes de l'EQE et du WPE en fonction de la densité surfacique de courant.

Il est également très utile d'avoir à disposition une caméra afin de mener une analyse qualitative rapide par exemple. Elle est souvent utilisée pour poser les pointes, mais peut servir à faire des mesures de résolution ou une analyse qualitative (défauts). On utilise ici la ThorLabs CS165CU - Zelux. Le site web de Thorlabs met à disposition des utilisateurs des SDK (Software Development Kit), entre autres pour ses caméras scientifiques. J'ai donc pu développer des fonctions qui permettent de lancer une vue en live, changer le temps d'exposition, le gain pour chaque couleur RGB ou encore l'Auto White Balance.

2 Spectro-imageur Rembrant

2.1 Description rapide de l'instrument

Un spectro-imageur permet, comme l'indique son nom, de faire à la fois de l'imagerie et de la spectroscopie. L'objectif avec cette outil est de pouvoir faire des cartographies de luminance sur des microdisplays sur ASIC. Pour pouvoir caractériser des micro-LEDs de plus en plus petites, la résolution spatiale demandée est de $2 \mu\text{m}$ pour un FOV de $8 \times 5 \text{ mm}^2$. Au vu de ces spécifications, le spectro-imageur devra faire plusieurs images afin de reconstituer le FOV en entier, ce qu'on appelle le stitching.

Côté spectroscopie, on a besoin d'une résolution spectrale qui peut aller jusqu'à 5 nm. Il nous sera également fourni un logiciel permettant l'analyse automatique des micro-écrans : uniformité, résolution, contraste, défauts, clignotement.

2.2 Descriptions des fonctions d'analyses automatiques fournies par ELDIM

Afin de préparer l'arrivée du spectro-imageur, un émulateur nous a été fourni afin d'avoir une première prise en main du pilotage par Python (voir l'image [4a]). Quelques une des fonctions proposées sont par exemple le redressement automatique de l'image, la correction de la distorsion et du vignettage, le stitching (création d'une image grand champ par combinaison de plusieurs prises de vues à champ réduit) ou encore l'autofocus. Pour chaque image en entrée, tous les pixels sont comptés, et les zones mortes ou surbrillantes sont localisées. Une fonction en particulier (le Mapping) permet d'analyser une image afin d'en faire ressortir tous les défauts : pixels, lignes ou colonnes mortes, pixels surbrillants etc. Ci-dessous une liste non-exhaustive de ces défauts :

- | | | |
|----------|----------------|----------|
| — LINE | — ROUND_CORNER | — BRIGHT |
| — COLUMN | — BLACK_SPOT | — GREY |
| — HBAND | — MONSTER_SPOT | — DEAD |
| — VBAND | — ASO | |

2.3 Exemples d'analyses

En attendant son arrivée donc, ELDIM nous a proposé l'émulateur suivant. Il permet une première prise en main en présentant déjà quelques fonctions de traitement d'image telles que le Mapping :

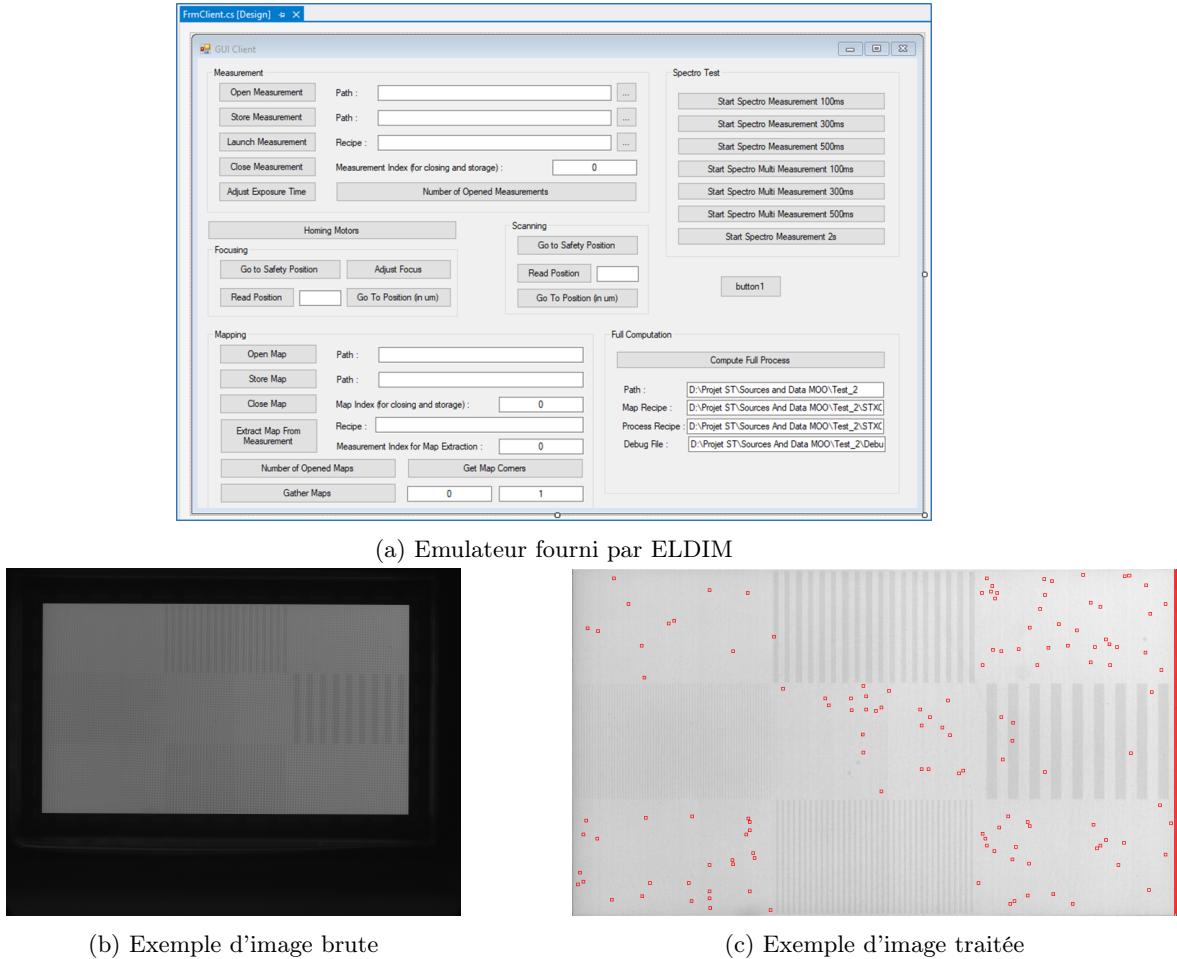
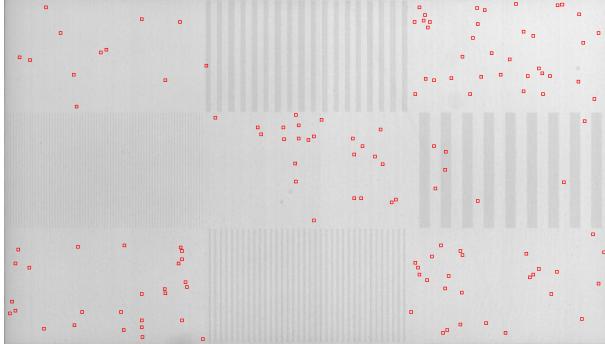


FIGURE 4 – Traitement d'une image réelle par l'émulateur

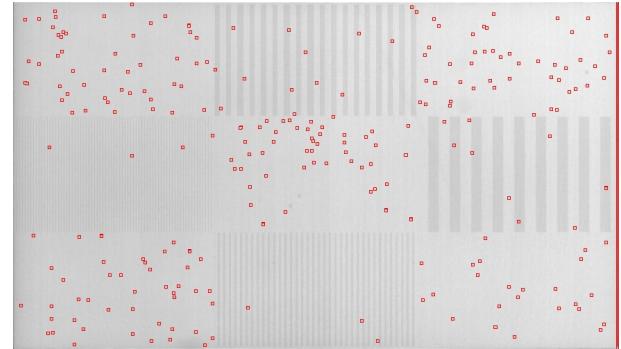
La détection de défauts est régie par différents paramètres permettant de contrôler la sensibilité. Lors du process de Mapping, il suffit de changer ces paramètres dans la MapRecipe. Ci-dessous, une liste de quelques-uns de ces paramètres :

- LinesAndColumns : Détection de lignes ou colonnes défectueuses
- ThresholdPercentage : Définit le seuil en pourcentage à partir duquel une ligne/colonne est considérée comme défectueuse
- RoundCorners : Détection de coins défectueux
- BlackAndMonsterSpots : Détection de défauts de types Black Spots ou Monster Spots
- LowThresholdRatio : Définit la limite en dessous de laquelle un pixel est considéré potentiellement "black". Si plusieurs pixels voisins sont détectés, un black spot est identifié
- HighThresholdValue : Définit la valeur maximale au-delà de laquelle un pixel peut être considéré comme "bright". Si plusieurs pixels voisins sont détectés, ils sont assimilés à un black spot
- AbnormalitySwitchOn : Détection de pixels anormalement allumés lors de l'acquisition noire
- ThresholdRatio : Définit la limite au-dessus de laquelle un pixel est considéré comme Abnormality Switch On (allumé)

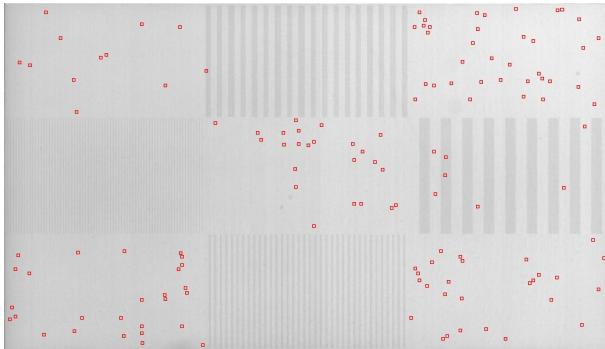
- BrightGreyDead : Détection de pixels morts, brillants ou trop éloignés en intensité par rapport à leurs voisins les plus proches
 - KernelSize : Définit la taille de la zone utilisée autour de chaque pixel pour le processus
- L'exemple ci-dessous montre comment la détection change effectivement en fonction de ces paramètres :



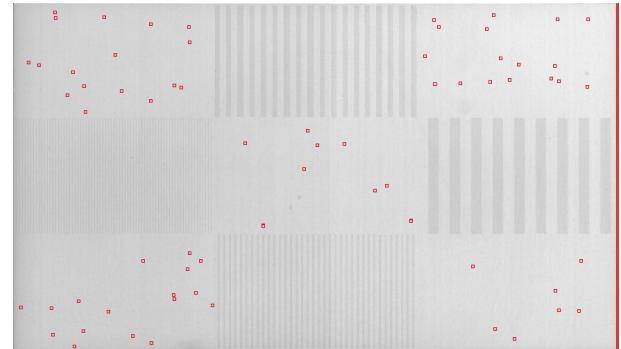
(a) BrightGreyDead : DeviationToleranceBright="10", DeviationTolerance="10", KernelSize="7"



(b) DeviationToleranceBright="5"



(c) DeviationTolerance="5"



(d) KernelSize="20"

FIGURE 5 – Évolution de la détection des défauts en fonction de certains paramètres

2.4 Emulateur et pilotage par Python

Les fonctions fournies par ELDIM étant écrites en C#, il nous faut une méthode permettant d'appeler la librairie avec Python. On fait pour cela appel à un fichier WSDL (Web Services Description Language), document XML décrivant la structure et les fonctionnalités d'un service web. Des modules tels que *zeep* ou encore *suds* sur Python permettent alors d'importer le fichier WSDL qui fait office de dictionnaire.

La première étape consiste à ouvrir le fichier *STXCom.exe* (cf Annexe 2[A.2]) en mode administrateur. Le traitement d'images peut ensuite se lancer via Python comme suit :

```

1 from zeep import Client
2 # Accès au WSDL et à l'adresse IP locale
3 client_soap = Client('Server_WSDL.xml')
4 client_soap.transport.session.proxies = {'http': 'http://localhost:8000/STXCom'}
5 # Exécution de certaines fonctions du logiciel
6 # Equivaut à cliquer sur Fetch Measurement après avoir renseigné Simu_BLACK_500_873.bmp dans l'onglet
    Path correspondant
7 print(client_soap.service.FetchMeasurement("C:\\\\[...]\\\\Images Test\\\\Simu_BLACK_500_873.bmp"))

```

La fonction *FetchMeasurement* est ici prise en exemple mais l'ensemble des fonctions de l'émulateur peuvent être appelées par la méthode, avec les mêmes paramètres en entrée. Dans le cas d'un Mapping, il faudrait par exemple préciser en plus l'adresse de la *MapRecipe* comme on l'aurait fait dans le soft. En supposant que *recipe* renvoie l'adresse de la *MapRecipe*, il suffit de rajouter les deux lignes suivantes en amont :

```

1 root = etree.parse(recipe)
2 print(etree.tostring(root))

```

La fonction permettant l'extraction de la Map black en simultanée de la Map colorée est alors :

```

1 print(client_soap.service.MapExtractionBlackAndRef(MeasureRefIndex, MeasureBlackIndex, MapRecipe)
2 print(client_soap.service.SaveMap(MeasureRefIndex, MeasurementRefPath)
3 print(client_soap.service.SaveMap(MeasureBlackIndex, MeasurementBlackPath)

```

Dans l'annexe 1 [A.1], vous trouverez la liste de toutes les fonctions prises en charge par l'émulateur.

2.5 Développement d'une bibliothèque pour piloter le déplacement d'une platine XYZ motorisée

Le spectro-imageur sera porté par une platine au dessus du Prober, permettant son déplacement motorisé en XYZ. En attendant son arrivée, j'ai eu l'occasion de travailler sur le contrôle du Integrated Step-Servo Motor par Python. Il s'agit d'une étape très importante car on doit pouvoir switcher facilement du spectro-imageur à la caméra du Prober. L'objectif est d'intégrer les scripts permettant le contrôle dans les librairies déjà existantes pour le contrôle du Prober.

Le positionneur est relié au PC par port série, permettant d'envoyer des commandes ligne par ligne. Les messages sont envoyés en bytes et l'ensemble des messages pris en charge par le positionneur sont recensés dans le fichier mentionné en annexe. J'ai développé la librairie *TSMlib.py* ainsi qu'une interface graphique pour le contrôle de la platine [6].

Les mouvements se font suivant les 3 axes XYZ, avec des distances relatives ou absolues, et soit en counts soit en μm . Une mire placée en dessous d'une caméra fixée sur le bras a permis de calibrer la valeur d'un count : $0,05 \mu\text{m}$ pour une résolution de 20.000 steps/rev. Il est possible de faire varier la résolution via la fonction 'EG' : Electronic Gearing.

Tous les mouvements se font en CW (ClockWise : -) et CCW (CounterClockWise : +). Au départ, le moteur a son offset à 0 dans le sens CCW pour les 3 axes. Tout mouvement se fait donc au début en CW. Pour un observateur positionné en face du bras donc : vers la gauche (axe1), vers soi (axe2) et vers le bas (axe3). Simon, le tout reste bloqué (cf Get Alarm). Dans ce cas, il suffit d'appuyer sur Reset Axis : le moteur est automatiquement ramené à sa position d'origine. Le reset se fait par le biais de la fonction FeedToSensor. Lorsque la limite gauche est atteinte, il faut choisir le Reset dans le sens CCW, qui est celui de départ. Les boutons ON/OFF agissent sur le moteur de l'axe choisi ; Home permet de ramener le moteur de l'axe choisi à sa position d'origine, ou encore les 3 moteurs si All Axis est coché ; Top fait remonter le bras jusqu'à sa hauteur maximale (aussi sa position d'origine, revient donc à faire Home avec Axis = 3) ; Go To permet de faire des déplacements relatifs/absolus en counts/ μm ; LM/LP (respectivement < 0 et > 0) correspondent aux limites que l'on peut imposer respectivement en CCW/CW, là aussi en pouvant choisir All Axis. Un dernier champ de texte juste en dessous de Clear Browser permet d'envoyer toute autre commande n'apparaissant pas sur l'interface graphique et pouvant être pris en charge par le moteur de la platine. J'ai ensuite rajouté une entrée de texte afin de pouvoir envoyer directement au format chaîne de caractère toute autre commande n'apparaissant pas sur l'interface.

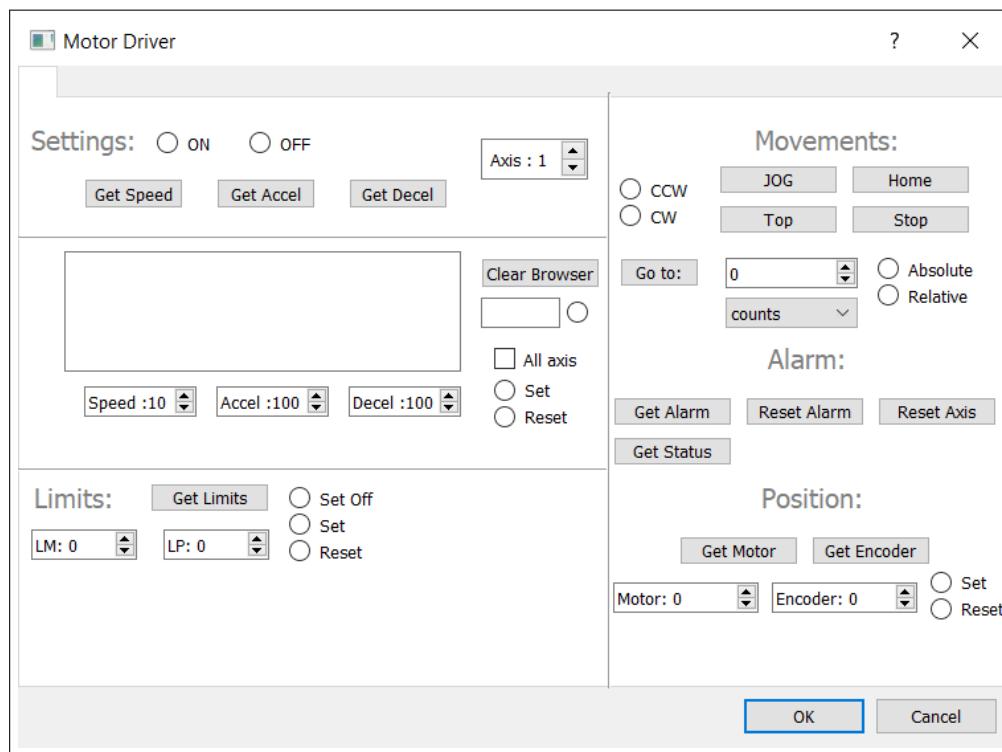


FIGURE 6 – Interface graphique pour le pilotage d'une platine motorisée

3 Traitement d'images

Cette troisième partie du document se penche sur le rôle central que joue le traitement d'images dans le processus de caractérisation des microécrans. En exploitant des techniques avancées, le traitement d'images permet de révéler des propriétés intéressantes des microécrans, telles comme le Signal Contrast font appel à des patterns bien spécifiques mais recensés dans le rapport d'activité mentionné en annexe.

3.1 Recensement de méthodes de caractérisation d'écrans

Cette section du document aborde les méthodes de caractérisation des écrans, englobant des paramètres tels que la luminance, le contraste, la résolution et la colorimétrie. Ces méthodes s'appuient sur les normes de l'IDMS (Information Display Measurements Standard) définies par l'ICDM (International Committee for Display Metrology). Les détails complets de ces mesures sont consignés dans le compte-rendu (CR) d'avril 2023, exposant en détail les protocoles appliqués et les résultats obtenus. L'adoption de l'IDMS garantit une évaluation impartiale et homogène des performances des écrans, conformément aux normes internationales.

1. Full-Screen : Mesure de la luminance (éventuellement les coordonnées trichromatiques et le CCT-Température de Couleur Corrélée) du centre d'un écran de certaines couleurs
 - White (Luminosité, Luminance de l'écran)
 - Black
 - Primary Colors
2. Peak luminance : Mesure de la luminance maximale atteinte par un carré (de par ex. $30 \pm 1 \text{ px}$) situé au centre de l'écran
3. Average peak luminance : Mesure du Peak White (centre de l'écran) puis de la luminance à au moins quatre autres points symétriques par rapport au centre. On fait ensuite la moyenne des mesures faites.
4. Mesure du contraste
 - Signal Contrast
5. Mesure du contraste des coins
6. Halo : Mesure de la luminance d'un petit rectangle noir sur un fond blanc. On dit qu'on observe des halos lorsque la luminance du fond blanc corrompt celle de rectangle noir.
7. Mesures électriques
 - Efficacité lumineuse : Rapport entre le flux lumineux d'un écran totalement blanc et la puissance électrique consommée
 - Efficacité de la luminance frontale : Rapport entre la luminance du Full-Screen White et la puissance électrique consommée
 - Efficacité de l'intensité frontale : Rapport entre l'intensité lumineuse du Full-Screen White et la puissance électrique consommée. Il suffit alors de multiplier l'efficacité de la luminance frontale par la surface de la LED.
8. Évaluation visuelle et échelles
 - Briggs Roam Test
 - Gray Scale : Pour une série de 9 (ou 17 ou encore plus) niveaux de gris, on mesure la luminance correspondante. Il suffit ensuite de tracer la variation de la luminance en fonction de niveau de gris.
- Color Gamma (Primary Color Scales) : Le même principe est appliqué sur les trois couleurs R,G et B.
9. Mesures spatiales
 - Réponse en fréquence spatiale (MTF) : Capacité du microdisplay à reproduire avec précision les variations de contraste à différentes échelles spatiales.
 - Résolution efficace : On considère cette fois-ci également la réponse spectrale de l'appareil de mesure de luminance
 - Pixels défectueux
10. Mesures temporelles
 - Temps de réponse : Temps mis par l'écran pour passer du noir au blanc puis revenir au noir
 - Warm-up : Temps mis par l'écran pour atteindre une luminance stable à $\pm 1\%$ (max 5%)
11. Mesures d'uniformité
 - Uniformité échantillonnée : Mesure de la répartition homogène des luminances dans un ensemble de points, calculée en faisant le rapport entre les valeurs maximale et minimale des luminances.
 - Uniformité de la zone : Évaluation de la répartition des luminances le long d'une coupe longitudinale dans une carte de luminance obtenue par un colorimètre d'imagerie, avec un positionnement de l'appareil à une distance définie.
12. Colorimétrie et Angle de vue
 - Color Gamut : Recherche de la plage des couleurs que le micro-écran peut reproduire
 - Visibilité angulaire : A l'aide d'un écran affichant une image totalement noire ou blanche. Il faut dans un premier temps mesurer le contraste C_θ en incidence normale ($\theta = 0$), puis aux angles d'incidence $\theta = \pm 15^\circ$, $\pm 30^\circ$ et $\pm 45^\circ$

3.2 Génération paramétrable d'images

Dans le but d'évaluer la conformité du spectro-imageur ainsi que l'efficacité du logiciel de traitement d'images, j'ai développé un code permettant de générer des images paramétrables. Ces images spécifiques vont servir de référence pour tester et valider les performances des équipements et des algorithmes d'analyse. J'ai utilisé deux modules clés : *PIL* (Python Imaging Library) et *OpenCV* (Open Source Computer Vision Library). L'utilisation de *PIL* a permis de créer, manipuler et exporter des images avec une flexibilité considérable. La bibliothèque offre des fonctionnalités étendues pour gérer des formats variés et pour ajuster précisément les paramètres visuels des images, adaptés aux besoins de la caractérisation. De manière complémentaire, *OpenCV* a joué un rôle crucial dans la manipulation d'images et l'implémentation d'algorithmes de traitement avancés. Cette bibliothèque, dédiée à la vision par ordinateur, a permis d'effectuer des opérations complexes telles que le filtrage, la détection d'objets et l'extraction de caractéristiques visuelles. Ces outils ont fourni la flexibilité et les fonctionnalités nécessaires pour mettre en œuvre les différentes fonctions explorées dans les parties ci-dessous. Toutes les fonctions que j'ai développées dans les parties suivantes ont également été recensées dans la partie annexe.

3.2.1 Résolution

L'objectif est de générer des images à des cycles/mm bien précis. La fonction prend en compte entre autres la définition de l'écran, sa taille, le pitch. Une seconde fonction permet ensuite de projeter l'image vers l'écran secondaire (ex : un micro-display).

Pour s'assurer que la fréquence de cycle obtenue est conforme aux attentes, deux méthodes distinctes sont mises en place pour la vérification. La première méthode s'appuie sur l'utilisation d'une caméra. En analysant les images projetées sur l'écran secondaire et en connaissant la taille des pixels de la caméra, il est possible de mesurer avec une grande précision la fréquence effective des cycles par millimètre sur l'écran. Cette approche visuelle permet d'obtenir une validation visuelle directe de la précision des cycles/mm générés. La seconde méthode de vérification fait appel à Python. Avant sa conversion en image, en projetant le tableau généré sur l'axe des abscisses on obtient un tableau à 1D que l'on appellera dans la suite *proj_bin*. On effectue ensuite un seuillage afin de simplifier les calculs. En accédant aux dimensions de l'image (w_{pix} le nombre de pixels) puis aux caractéristique de l'écran (notamment le pitch), on se retrouve avec la taille de l'image sur l'écran w_{ecran} . On a ici le choix :

- Une simple boucle sur cette même liste *proj_bin* permet en outre d'avoir accès à la période : appelons la $2 * c$. Il suffit alors de diviser la longueur totale de la liste *proj_bin* par $2 * c$ pour avoir le nombre total de paires de lignes, puis encore par w_{ecran} pour obtenir le nombre de paires de lignes/mm.
- En utilisant la fonction Counter, on a ensuite accès au nombre d'occurrences de chaque élément dans la liste *proj_bin*, donc le nombre de 0 et 1. Le nombre de lignes est cette fois ci obtenu en divisant le nombre d'occurrences de 1 (*Counter(proj_bin)[1]*) par cette fois-ci c

Dans le cas de l'image [7], l'écran utilisé en était un de 2,5 pouces de Microoled, d'où les paramètres suivants :

```

1   pitch, taille_ecran, definition, w_ecran, h_ecran = 5E-3, 0.38, (1746, 1000), 8.73, 5
2   freq = 1

```

On peut d'ailleurs en théorie remonter à la fréquence maximale pouvant apparaître sur l'écran avec comme seule information la taille des pixels sur celui-ci : il s'agit du cas où chaque paire de ligne occupe deux pixels de suite. Cela n'est cependant valable que dans le cas où les lignes sont parfaitement droites et bien alignées.

3.2.2 Détection des défauts

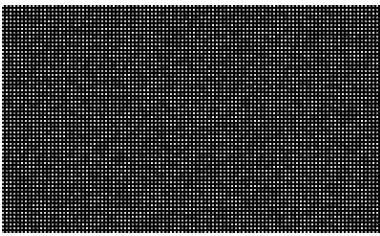


FIGURE 8 – Image fictive de pixels aléatoirement brillants

On cherche ici à vérifier si le process de traitement d'images fourni par ELDIM recense bien tous les types de défauts demandés. La fonction utilisée peut générer des pixels plus ou moins brillants (cf. image [8]), puis des lignes, colonnes et pixels morts ainsi que des BlackSpots plus ou moins grands. Pour chaque cas, des coordonnées aléatoires sont générées et le code vérifie qu'il s'agit bien d'un coin supérieur gauche par exemple. Le cas échéant, il vérifie que le BlackSpot rentre également bien dans l'espace qui le sépare des bords puis le crée aux dimensions demandées. On introduit ensuite un tilt en degrés puis on colle l'image sur un fond noir à des dimensions bien précises : 7904×6048 intrinsèques à la caméra. Ci-dessous, nous avons l'exemple d'une image avec une ligne et une colonne ainsi que quelques pixels morts. L'image au centre présente également un tilt. On voit que tous les défauts sont effectivement détournés et l'image est recentrée. Le tilt semble cependant ne pas avoir disparu entièrement. Il s'agit d'un soucis des fonctions fournies par ELDIM

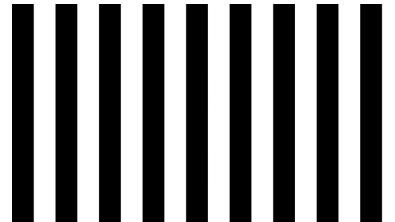
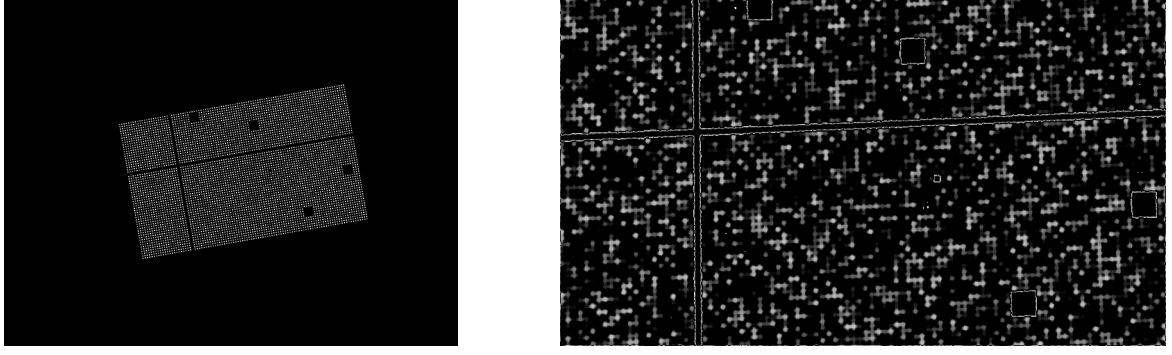


FIGURE 7 – Image d'une mire à 1 pl/mm

et les démarches ont déjà été entamées avec eux afin de trouver une solution puisqu'on n'a pas accès aux fonctions qu'ils utilisent pour faire le Mapping.



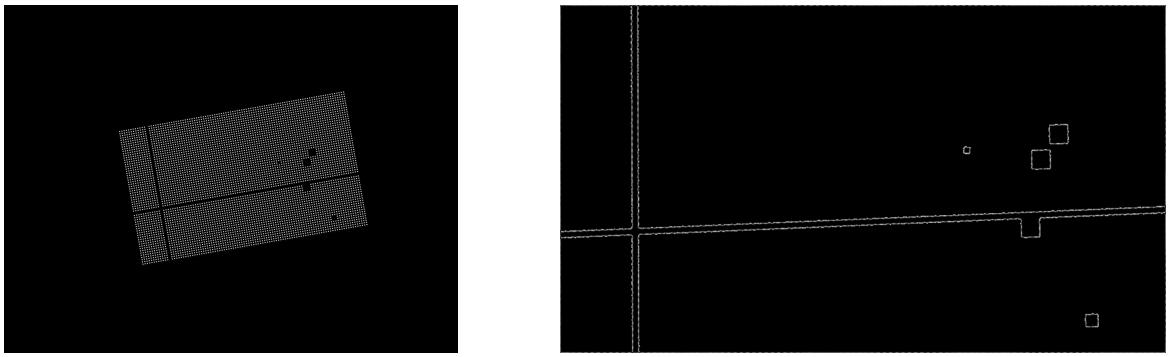
(a) Exemple d'image générée

(b) La même image traitée

FIGURE 9 – Traitement d'une image fictive par un émulateur

En faisant varier l'orientation de l'image, on remarque en outre que l'émulateur ne fonctionne pas lorsque l'image ne présente aucun tilt. Il faut également que l'image ait impérativement un fond noir et des dimensions spécifiques mais également une profondeur de bits de 8.

Il est possible de générer l'image sans aucune boucle (utile si on veut des images encore plus grandes), le prix à payer étant que l'on a que des pixels noirs ou blancs. Si on veut des pixels aléatoirement brillants, il faut forcément parcourir tous les pixels de l'image. On n'a pas cette contrainte pour des pixels noirs et blancs car c'est un même modèle que l'on reproduit sur toute l'image, ce qui est possible sans boucle apparente et en utilisant uniquement des fonctions de Numpy tels que repeat, concatenate, tile etc. Dans ce dernier cas, on n'observe que les défauts entourés sur un fond noir comme sur l'image [10b], contrairement à l'image [9b] où on voit ce qui semble être les pixels surbrillants. On voit d'ailleurs encore mieux sur cet exemple le défaut d'effacement du tilt par le soft, s'agissant peut-être d'un problème de rognage. Même avec un tilt de 1 degré, l'image est cependant traitée.



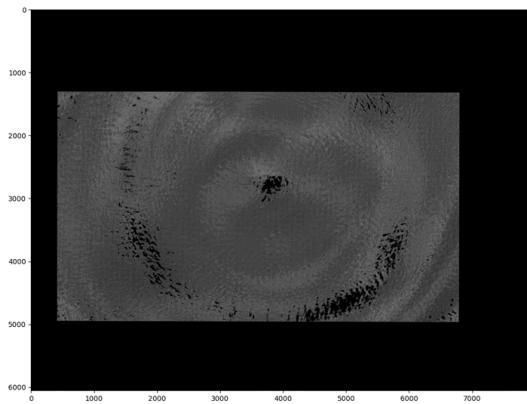
(a) Image générée en noir et blanc

(b) La même image traitée

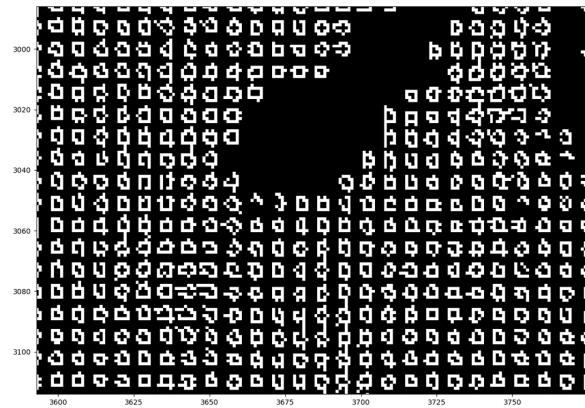
FIGURE 10 – Traitement d'une image (pixels noirs et blancs) par un émulateur

3.3 Développement d'un programme d'analyse d'image

Certaines fonctions, en particulier du module OpenCV sur Python, permettent l'identification ainsi que le détournage des contours sur une image. J'ai donc cherché à les utiliser sur les exemples d'images réelles fournis par ELDIM. Une de ces fonctions, appelée Canny, permet d'accomplir cette tâche et c'est en l'associant à un script permettant de détourer ces mêmes contours que j'ai obtenu l'image [11a]. En zoomant assez, on voit ce qui semble être des pixels très ordonnés mais avec des formes très irrégulières. C'est également le cas de la zone morte mise en évidence. Cette irrégularité a rendu difficile l'exploitation de ces données malgré la tentative d'utilisation de détecteurs de cluster/blob ou encore le calcul de pitch moyen pour être sûr d'être au centre du pixel et ainsi traverser l'image pixel par pixel. La méthode semble donc pouvoir bien détourer la zone centrale mais beaucoup de mal avec les pixels comme on peut le voir au niveau des zones mortes. Ces dernières n'apparaissent d'ailleurs pas lorsque la même image est traitée avec le soft de chez ELDIM. Je continuerai mes recherches sur les méthodes qui existent pour contourner les difficultés rencontrées dans cette partie.



(a) Image traitée par la fonction Canny



(b) Zoom sur une zone morte

FIGURE 11 – Traitement de l'image Green avec Python

Perspectives

Les objectifs à atteindre au cours des prochains mois sont les suivants :

- Intégrer la librairie permettant le contrôle du support motorisé dans le programme central de contrôle du Prober
- Continuer et tester le développement de programmes d'analyse d'images
- Intégrer le spectro-imageur au Prober TS-3500
- Avancer dans l'automatisation des mesures sur ce même Prober

A Annexes

A.1 Fonctions proposées par l'émulateur fourni par ELDIM

• AdjustExposureTime	• GetMeasurementPosition	• ProcessAverageAndDevMap
• AdjustSpectroExposureTime	• GetMeasurementsCollection	• ProcessMap
• DefineExposureTime	• GetScanningPosition	• ProcessingCompleted
• DefineSpectroExposureTime	• GetSystemInformation	• ResetCameraMemory
• FetchMap	• GetSystemStatus	• SaveMap
• FetchMeasurement	• GoToSafetyPosition	• SaveMeasurement
• FlushMap	• GoToScanningSafetyPosition	• SetFocusPosition
• FlushMeasurement	• HomingMotorPosition	• SetMeasurementPosition
• FreeMapExtraction	• MapCorners	• SetScanningPosition
• FreeMapExtraction-BlackAndBToRG	• MapExtraction	• StartMeasurement
• FreeMapExtractionBlackAndRef	• MapExtractionBlackAndBToRG	• StartSpectroLoop
• FreeMapExtractionToRef	• MapExtractionBlackAndRef	• StartSpectroMeasurement
• FreeStartMeasurement	• MapExtractionToRef	• StartSpectroMeasurementDark
• Gather2Maps	• MappingCompleted	• StartSpectroMeasurementDarkLinear
• GetErrorCode	• MeasurementCompleted	• StartSpectroMeasurementLinear
• GetErrorText	• MeasurementCorners	• StopSpectroLoop
• GetFocusingContrast	• PerformSpectroCalibration	• StreamMap
• GetInterfaceRevision	• PreProcessMap	• StreamMeasurement
• GetMapsCollection	• ProcessAverageAndDev-CenterMeasurement	

A.2 Lien vers les codes Python utilisés

- Fonction de contrôle de l'émulateur fourni par ELDIM : [test_zEEP.py](#)
file:S:/230-Composants_Optiques_Multimédias/230.32-Composants_Emissifs/1-Instrumentation/14-Banc_Rembrant/EmulateurSpectroImageur/TestWSDLPython
- Logiciel à lancer en mode administrateur lors de traitements d'images : [STXCom.exe](#)
file:S:/230-Composants_Optiques_Multimédias/230.32-Composants_Emissifs/1-Instrumentation/14-Banc_Rembrant/EmulateurSpectroImageur/SetupPremierstests/service/bin
- Logiciel de génération paramétrable d'images : [generate.py](#)
file:S:/230-Composants_Optiques_Multimédias/230.32-Composants_Emissifs/1-Instrumentation/14-Banc_Rembrant/EmulateurSpectroImageur/Générationd'images
- Compte-rendu de méthodes de caractérisations d'écrans : [CR activité_Avril2023.pdf](#)
file:S:/120-LETI/120.3-DOPT/120.3.4-SNAP/120.3.4.8-LITE/Stages-Alternants/2022/TallaDIAGNE/Rapports
- Toutes les fonctions utilisées pour le contrôle du SMU, de la caméra et du spectro :
[file:S:/230-Composants_Optiques_Multimédias/230.32-Composants_Emissifs/1-Instrumentation/14-Banc_Rembrant/Common-SMUetcaméra](#)
- Librairie et interface permettant le contrôle du support motorisé du spectro-imageur :
[file:S:/230-Composants_Optiques_Multimédias/230.32-Composants_Emissifs/1-Instrumentation/02-Prober_TS3500/7-Instrumentation/CommandeDuPositioneurMPI/PythonLibrary/TSM_motor_driver-main/PythonComport](#)
- Document recensant toutes les commandes prises en charge par ce moteur :
[Host-Command-Reference_920-0002W_0.pdf](#)
file:S:/230-Composants_Optiques_Multimédias/230.32-Composants_Emissifs/1-Instrumentation/02-Prober_TS3500/7-Instrumentation/CommandeDuPositioneurMPI/DrivermoteurTSM17Q-1RG