

DUT MESURES PHYSIQUES

**Participation au montage d'une expérience de microscopie de  
fluorescence mettant en œuvre des principes d'holographie  
numérique pour le suivi de nanoparticules**

Kaiyuan XU

Étudiant en thèse :

Florian SEMMER

Tuteur laboratoire :

François MARQUIER

Tuteur IUT :

Fabienne GOLDFARB

# TABLE DES MATIÈRES

<b>1. INTRODUCTION</b>	<b>3</b>
<b>1.1. PRESENTATION DE L'EQUIPE DE RECHERCHE</b>	<b>3</b>
<b>1.2. DESCRIPTION DU MONTAGE EXPERIMENTAL</b>	<b>3</b>
1.2.1. PRINCIPE DU TRACKING DE PARTICULE	3
1.2.2. PRINCIPAUX ELEMENTS ET CARACTERISTIQUES	4
<b>1.3. OBJECTIF DE MON TRAVAIL</b>	<b>8</b>
1.3.1. AIDER A LA CONSTRUCTION D'UNE INTERFACE HOMME-MACHINE POUR	8
1.3.2. CONTRAINTES	8
1.3.3. OBJECTIF CONCRET	9
<b>2. SIMULATIONS D'EXPERIENCES</b>	<b>9</b>
<b>2.1. PRINCIPE ET OBJECTIF DES SIMULATIONS</b>	<b>9</b>
2.2. MISE EN APPLICATION : INTERFACE LABVIEW/PYTHON	10
2.3. AFFICHAGE LABVIEW D'UNE IMAGE DE MICROSCOPIE OBTENUE GRACE A UN PROGRAMME PYTHON	11
2.3.1. FONCTIONS DE BASE	11
2.3.2. PRISE EN COMPTE D'UNE FONCTIONNALITE DE ZOOM	12
2.4. MISE EN APPLICATION : DETERMINATION DE LA POSITION PRECISE DES NANOPARTICULES	13
2.4.1. AJUSTEMENTS GAUSSIENS	13
2.4.2. PRECISION DE LOCALISATION	14
<b>3. EXPERIENCE</b>	<b>16</b>
<b>3.1. PILOTER LA PLATINE PIEZOELECTRIQUE</b>	<b>16</b>
3.1.1 PRINCIPE DU SCAN PIEZO ELECTRIQUE :	16
3.1.2 CODE LABVIEW A REALISER	16
<b>3.2. OBTENTION DE LA PREMIERE IMAGE</b>	<b>18</b>
<b>3.3. OBTENTION DE LA DEUXIEME IMAGE (ZOOM)</b>	<b>19</b>
3.3.1 LE PRINCIPE DU SCAN EN HOLOGRAPHIE NUMERIQUE	19
3.3.2 CODE LABVIEW A REALISER	20
3.3.3 OBTENTION DE LA DEUXIEME IMAGE	20
<b>4. CONCLUSION</b>	<b>20</b>
<b>5. RÉFÉRENCES</b>	<b>22</b>
<b>6. ANNEXES</b>	<b>23</b>
<b>LIST OF COMMANDS FOR THE USB INTERFACE</b>	<b>38</b>

## 1. Introduction

### 1.1. Présentation de l'équipe de recherche

Mon stage s'est déroulé dans le groupe « biophotonique et physiopathologies des synapses » du laboratoire Lumière Matière et Interfaces (LuMIn) à l'Ecole Normale Supérieure Paris-Saclay. Le laboratoire s'intéresse à l'interaction lumière-matière à différentes échelles (atomes, matériaux, dispositifs, systèmes vivants) et ses applications multidisciplinaires. Il propose des synergies nouvelles et originales aux frontières de l'optique et de la physique quantique, des technologies des dispositifs, ainsi que l'exploration in vitro et in vivo de processus biologiques fondamentaux pour une meilleure compréhension de la pathogenèse des cancers et des maladies du cerveau.

L'activité principale de ce laboratoire repose sur un large spectre de compétences en optique (lasers, optique non linéaire, physique quantique, plasmonique), avec des développements applicatifs pour la conception et l'élaboration de matériaux, dispositifs micro- et nanophotoniques, circuits microfluidiques, et pour l'étude des phénomènes biochimiques dans les cellules, les tissus et les organismes vivants. Ce programme de recherche pluridisciplinaire et multi-échelles aborde des questions sociétales majeures telles que le traitement et le stockage de l'information, le développement durable et les sources d'énergie alternatives, et la santé publique.<sup>[1]</sup>

Mon travail, encadré par Florian Semmer (étudiant en thèse) et François Marquier (enseignant-chercheur), a été relié à la construction d'un système de microscopie à deux photons pour le suivi de nanoparticules en mouvement. La spécificité de l'expérience réside dans l'utilisation d'hologrammes pour exciter la luminescence de marqueurs biologiques dans des neurones. L'idée est de suivre rapidement la nano-particule dans son mouvement le long de l'axe optique (z) ou dans un plan orthogonal à cette direction (x,y). Les hologrammes ont déjà été utilisés pour faire de l'imagerie bi-photon<sup>[2] [3]</sup> et nous voulons aller plus loin en proposant un affichage dynamique des hologrammes s'adaptant à la luminescence collectée afin de suivre une particule mobile dans une tranche de cerveau.<sup>[4]</sup>

### 1.2. Description du montage expérimental

#### 1.2.1. Principe du tracking de particule

L'objectif de l'expérience que l'équipe souhaite monter est de suivre en temps réel le déplacement d'une nanoparticule brillante. On doit pour cela focaliser un laser sur la particule (elle émet alors de la lumière par fluorescence ou par génération de second harmonique suivant le type de nanoparticule et/ou de laser utilisés) : on veut donc que le spot laser suive la particule quand elle bouge. Pour cela, nous utilisons un système d'holographie numérique, affiché à l'aide d'une matrice de micro-miroirs (DMD pour Digital Micromirror Device). Le dispositif permet d'imprimer sur le DMD un hologramme qui modifie la phase du faisceau laser incident et déplace le spot laser dans le plan de l'échantillon. Le schéma de principe de l'expérience est décrit sur la Figure 1.

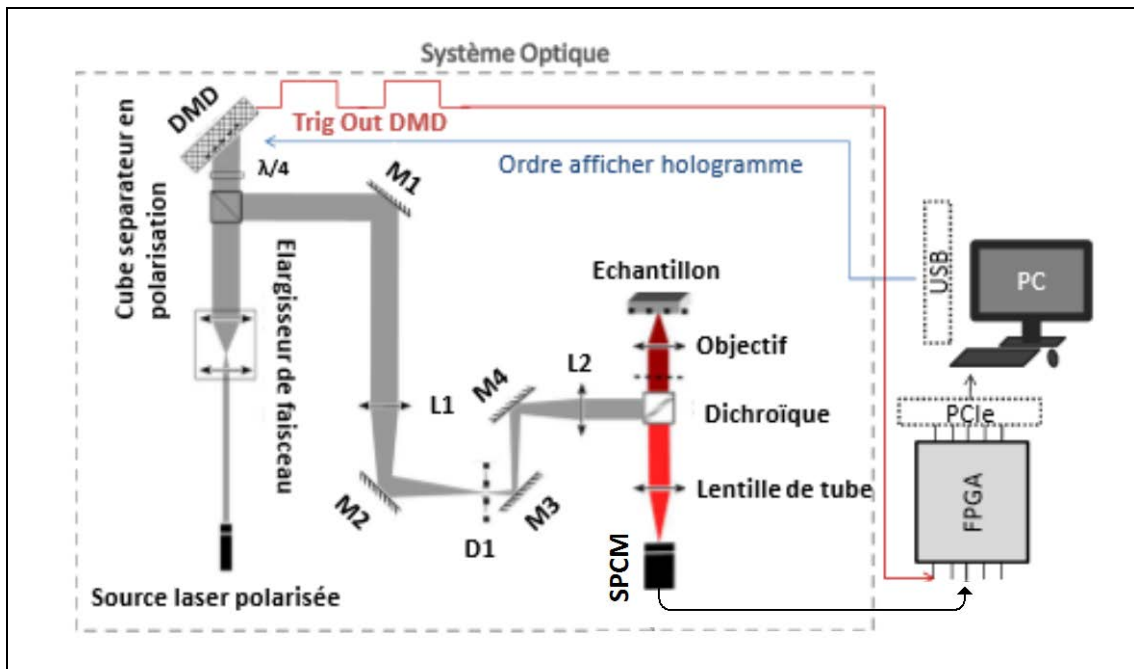


Figure 1 : Schema de principe de l'expérience de tracking de particules.

Schéma du montage. L1 (  $f_1 = 300 \text{ mm}$  ) projette la transformée de Fourier de l'hologramme dans son plan focal image où l'on applique un filtre passe-bande avec le diaphragme D1. Ensuite on projette l'ouverture de D1 sur l'échantillon à l'aide de L2 (  $f_2 = 500 \text{ mm}$  ) et l'objectif (  $f_{obj} \approx 3.3 \text{ mm}$  ). La lumière émise dans l'échantillon est séparée de la lumière excitatrice par un dichroïque et collectée sur le détecteur (compteur de photon) grâce à la lentille de tube (  $f_{tube} = 200 \text{ mm}$  ). On s'aligne sur l'ordre zéro de diffraction par la matrice de miroir que l'on sépare de la lumière incidente avec le couple lame à retard quart d'onde et cube séparateur en polarisation. Le signal du détecteur ( Single Photon Counting Module SPCM ) est envoyé vers le FPGA, codé pour compter les fronts montants lorsque le DMD est actif, on peut ainsi compter les photons émis par la particule. De cette manière l'ordinateur peut associer un hologramme au nombre de photons émis<sup>[5]</sup>

### 1.2.2. Principaux éléments et caractéristiques

#### Hologramme

Un hologramme est historiquement un objet diffractant construit à partir des interférences entre la lumière d'un laser (faisceau de référence) et celle diffusée par un objet éclairé par le même laser (faisceau objet). Si la lumière diffusée par l'objet est coupée, la diffraction du faisceau de référence sur l'hologramme permet de retrouver l'information en amplitude et en phase de la lumière diffusée.

Dans notre expérience, la figure d'interférence produisant l'hologramme est en fait calculée en fonction de la position d'un objet virtuel, puis envoyée au DMD. On affiche l'hologramme noir et blanc avec les micro-miroirs du DMD qui peuvent s'orienter selon un angle de  $+12^\circ$  ou  $-12^\circ$ . La figure 2 représente un hologramme : les deux couleurs sont associées aux deux orientations possibles des miroirs. Le disque qui apparaît est la zone éclairée par le laser. Le DMD présente des défauts de planéité que l'équipe corrige avec l'algorithme de génération d'hologrammes. Les franges jouent le rôle d'un réseau de diffraction pour lequel les

différents ordres portent une information sur l'objet virtuel. En modifiant ce réseau de diffraction, on modifie la direction et la phase portée par le laser ce qui conduit à une modification de la position du spot après focalisation. <sup>[6]</sup>

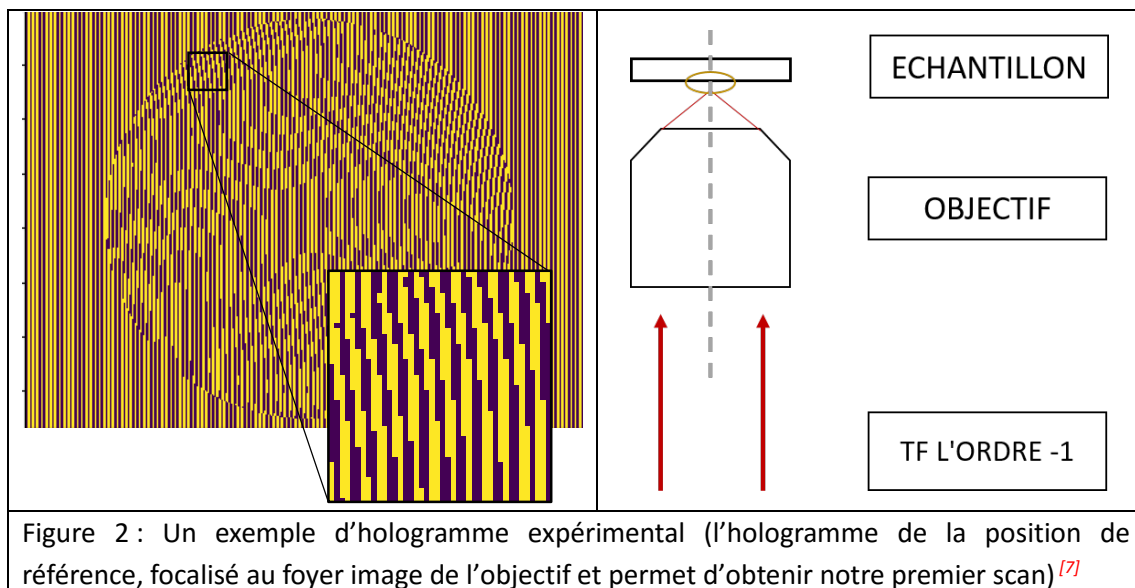


Figure 2 : Un exemple d'hologramme expérimental (l'hologramme de la position de référence, focalisé au foyer image de l'objectif et permet d'obtenir notre premier scan) <sup>[7]</sup>

#### Matrice de micro-miroirs (Digital Micromirror Device DMD)

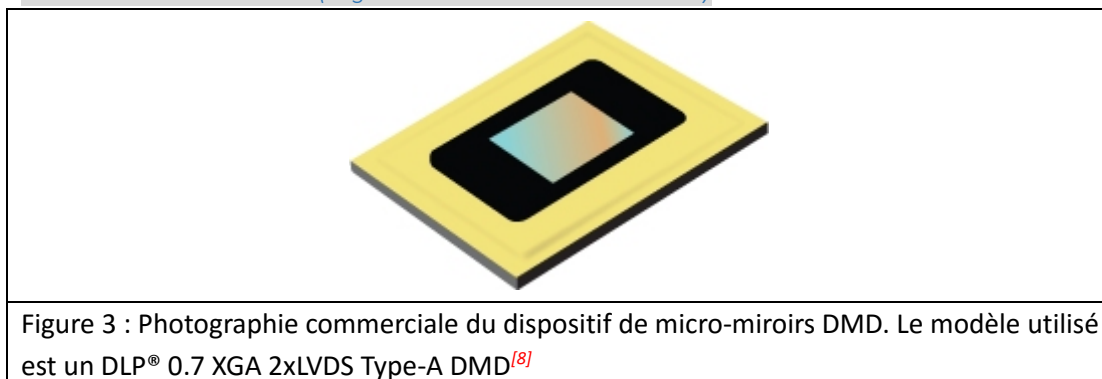


Figure 3 : Photographie commerciale du dispositif de micro-miroirs DMD. Le modèle utilisé est un DLP® 0.7 XGA 2xLVDS Type-A DMD <sup>[8]</sup>

Un DMD est une matrice de miroirs carrés de dimension micrométrique (cf. Figure 4). Chaque miroir peut pivoter autour d'une de ses diagonales et être dans deux positions différentes à plus ou moins 12 degrés par rapport à la normale au plan moyen du DMD.

Lorsque le DMD n'est pas alimenté, les miroirs sont tous dans le plan moyen de la surface. Lorsqu'il est alimenté, les miroirs s'orientent et on peut envoyer une image au dispositif. L'image est binaire, un pixel pouvant être noir ou blanc. Lorsqu'un pixel est « noir », l'angle d'orientation du miroir correspondant  $\theta_m = +12^\circ$  comme schématisé figure 4.

Le DMD que nous utilisons ici est constitué de  $768 \times 1024$  pixels de taille de côté :  $d = 13,6 \mu\text{m}$ .

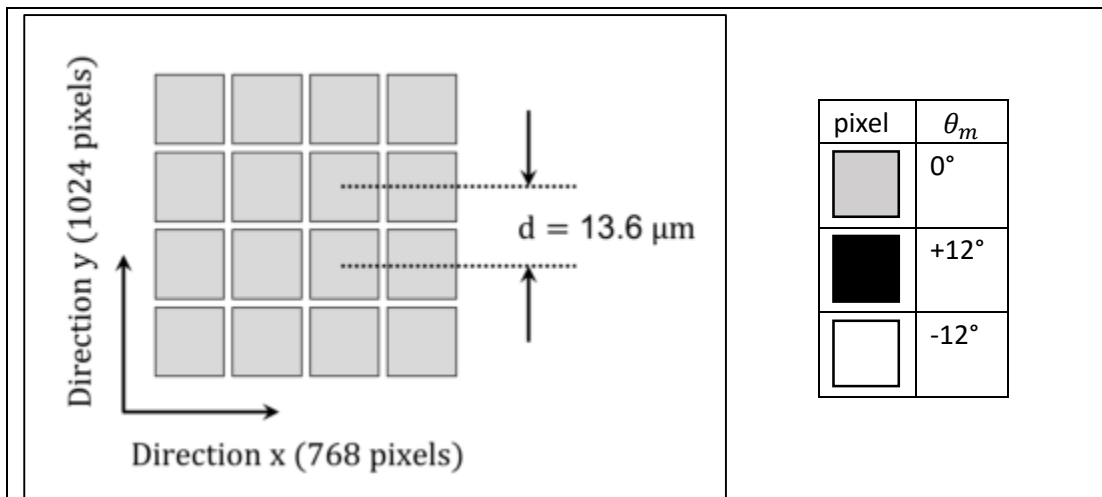


Figure 4 : Description du DLP7000

A gauche : La matrice de micromiroirs est constituée de 1024 x 768 pixels, dans les directions y et x respectivement. La période  $d$  du réseau de miroirs est de  $13.6 \mu\text{m}$ .

A droite : La représentation d'angle avec la couleur

Les points rouges dans le cas n°1 sont une représentation d'une figure de diffraction dans le plan de Fourier (c'est-à-dire au foyer d'une lentille) par le DMD d'un faisceau laser étendu lorsque le DMD n'est pas alimenté (orientation des miroirs à  $0^\circ$ ). Maintenant, on fait varier l'orientation des miroirs, on choisit l'angle d'orientation soit  $+12^\circ$  soit  $-12^\circ$  pour afficher des hologrammes. Alors on obtient une figure d'illumination en rouge, comme le cas n°2 à droite à l'aide d'un réseau de période  $4d$ . Dans notre expérience, on envoie l'ordre -1 vers le microscope car cet ordre est le plus intense.

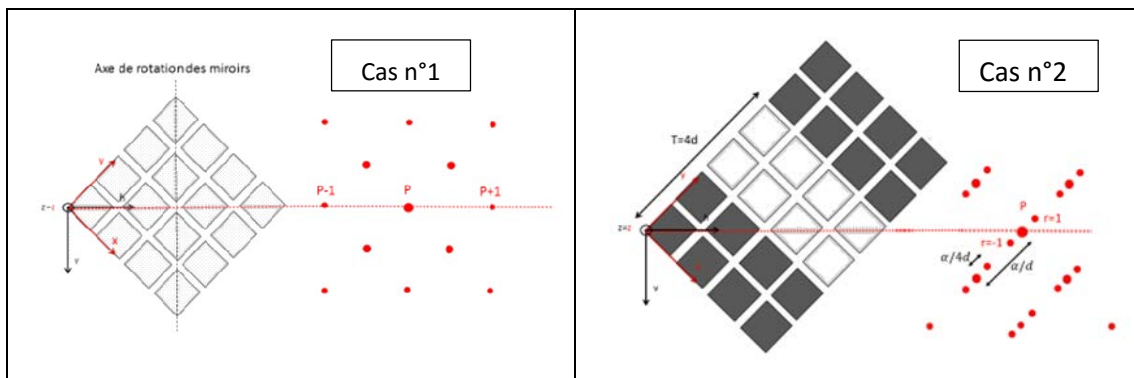


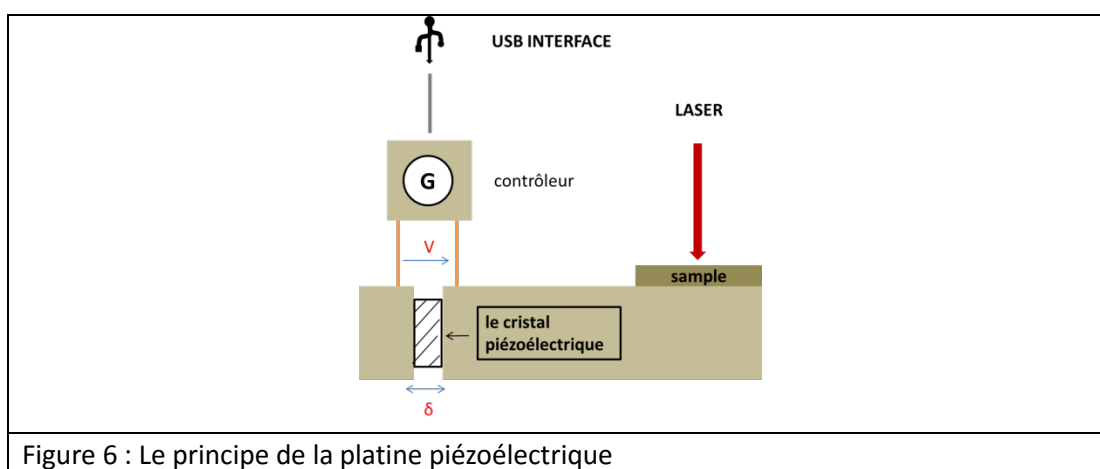
Figure 5 : Figure d'illumination de DMD

Cas n°1 à gauche Figure d'illumination due à la diffraction par les miroirs. Les miroirs sont comme des pixels avec lesquels on peut afficher des hologrammes. Lorsqu'on modifie les orientations des miroirs, on peut sur-imprimer un motif (ici un réseau de période  $4d$ ). On obtient alors une figure d'illumination comme le cas n°2 à droite où les miroirs blancs ont orientés d'un angle  $-12^\circ$  et les miroirs noirs  $+12^\circ$ .<sup>[9]</sup>

### Platine piézoélectrique

Dans notre expérience, on utilise une platine piézoélectrique de translation pour déplacer le porte-échantillon. La piézoélectricité est la propriété que possèdent certains matériaux à se

polariser électriquement sous l'action d'une contrainte mécanique et réciproquement de se déformer lorsqu'on leur applique un champ électrique. Quand on augmente la tension ( $V$ ) aux bornes d'un tel cristal, le déplacement ( $\delta$ ) augmente aussi (voir Figure 6).<sup>[10]</sup>



La platine que nous utilisons est une platine piezoconcept LT3 (Figure 7). C'est une platine à 3 axes (X, Y, Z) avec une plage de mouvement allant jusqu'à 200  $\mu\text{m}$  sur X et Y et Z. Une électronique de détection de position et un contrôle de rétroaction proportionnel et intégral (PI) en boucle fermée permettent une grande précision de positionnement.<sup>[11][12]</sup>



Figure à gauche : LT3.200 PIEZOCONCEPT

Figure à droite : 3 Axis Controller PIEZOCONCEPT, commandable en Labview en utilisant la 'USB interface'

Nous verrons dans les sections suivantes que la platine est utilisée pour faire un scan que nous appelons 'Scan « grossier » piezo'. Après avoir obtenu notre premier scan, on va aussi l'utiliser pour déplacer le porte-échantillon à une position choisie (détaillée à la partie 3 "expérience") avant de faire un deuxième scan, cette fois à l'aide du DMD.

### Détecteurs (Comptage de photons)

Le comptage de photons est une technique dans laquelle chaque photon est compté en utilisant un détecteur très sensible (SPCM pour Single Photon Counting Module).<sup>[15]</sup> Nous l'utilisons pour mesurer l'intensité émise par les nanoparticules à des positions données. Le

modèle de compteur de photons que nous utilisons est le SPCM-AQRH-TR (Figure 8), qui possède un bruit de base de 1000 photons/seconde (« dark count »).



Figure 8 : SPCM-AQRH-TR<sup>[16]</sup>

#### Carte FPGA

Les réseaux de portes programmables sur site (FPGA) sont des circuits intégrés reprogrammables qui contiennent un ensemble de blocs logiques programmables. L'adoption des puces FPGA est motivée par leur flexibilité, leur rapidité et leur fiabilité en fonction du matériel, et leur capacité à gérer des opérations parallèles.<sup>[17]</sup>

Pendant notre expérience, on compte les photons à une fréquence de 80 MHz. La cadence nous pousse à choisir une solution d'acquisition rapide, d'où la carte FPGA, associée au module LabVIEW correspondant.

#### microscope

On utilise un microscope inversé pour la simplification d'accès à l'échantillon.

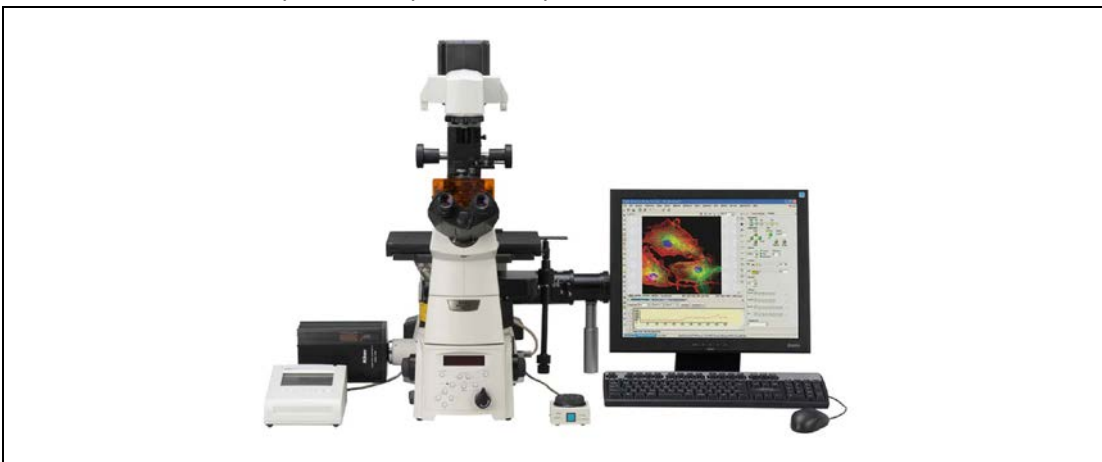


Figure 9 : microscopes inversés Eclipse Ti Series<sup>[18]</sup>

### 1.3. Objectif de mon travail

1.3.1. Aider à la construction d'une interface homme-machine pour Piloter l'expérience

Aider à faire les alignements

1.3.2. Contraintes

Utilisation de python pour les algorithmes de suivi



Utilisation de labview pour les interfaces homme-expérience

### 1.3.3. Objectif concret

Coupler Python et Labview

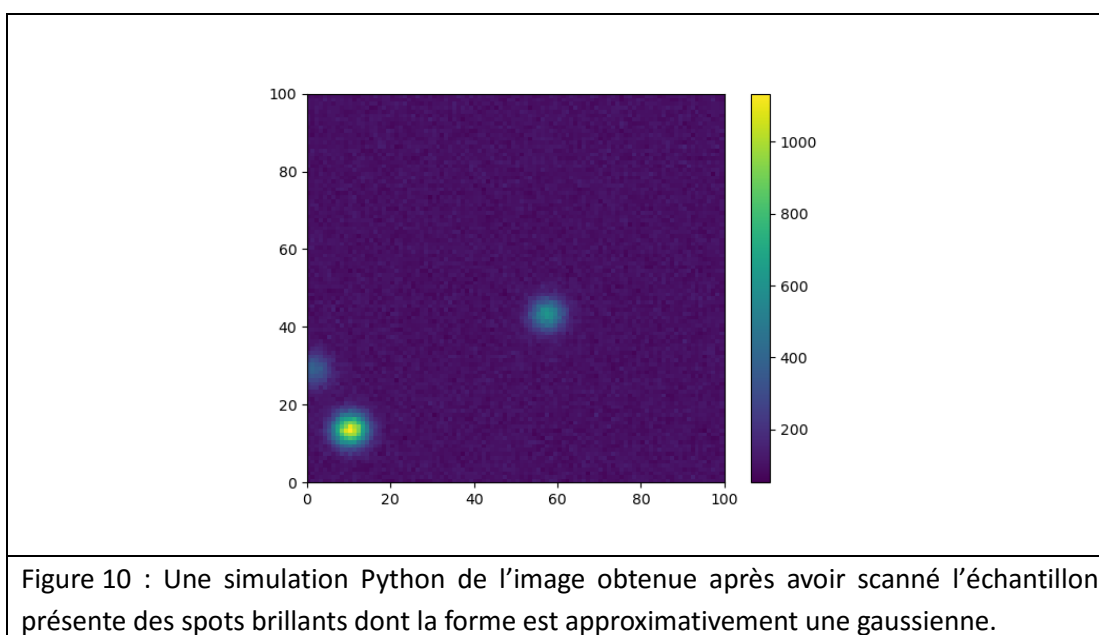
Aider à la structuration de l'interface

## 2. Simulations d'expériences

Le stage a débuté le 11 mai mais je n'ai pu venir au laboratoire que mi-juin. Le travail s'est donc partiellement fait à distance. Une partie importante de ce que j'ai réalisé est donc reliée à des simulations d'expérience, qui ont permis d'avancer sur des points techniques de l'interface Labview-Python, ou bien des fonctionnalités Labview utiles qu'il était possible de développer sans l'expérience.

### 2.1. Principe et objectif des simulations

Dans l'expérience, on repère à l'aide d'un microscope des nanoparticules qui émettent de la lumière. L'image obtenue après avoir scanné l'échantillon présente des spots brillants dont la forme est approximativement une gaussienne. La Figure 10 représente ce type d'image simulée informatiquement.



En l'absence d'accès à l'expérience, nous avons simulé des images avec des positions de particules fixées. Les programmes sont écrits avec deux configurations, correspondant à deux types d'acquisition différents :

*main1 - python.py* : simule le résultat d'un scan sur une « grande » surface de l'ordre de 35 micromètres de côté (qui sera à terme effectué par la platine de translation piézo commandable facilement mais lente) (**cf. ANNEXE 1**)

*main2 - python.py* : simule le résultat d'un scan réalisé avec des hologrammes, plus rapide et plus précis sur une zone plus petite, de l'ordre de 2 micromètres de côté (**cf. ANNEXE 2**)

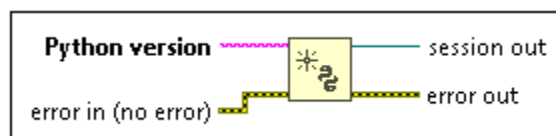
Les paramètres importants de ces programmes sont le nombre de pixels (npx), la taille d'un pixel en nanomètre (d) et, pour le 2<sup>e</sup> programme seulement, les coordonnées du centre de l'image (X\_en\_um et Y\_en\_um).

## 2.2. Mise en application : interface Labview/python

L'algorithme de tracking des nanoparticules demande des calculs complexes et une interaction avec le DMD qui étaient déjà codés en python. L'interface avec l'utilisateur devait être programmée en Labview, il fallait donc dans un premier temps mettre en place le dialogue entre ces deux langages.

Il existe trois fonctions Python dans LabVIEW pour organiser l'interaction :

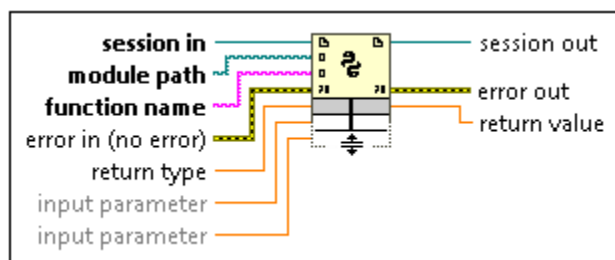
- Ouverture d'une session python



Ouvrir une session Python

Cette fonction ouvre une session python avec une version spécifique. L'ouverture de session est nécessaire pour les exécutions de fonctions python ensuite

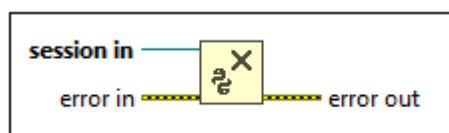
- Utilisation de fonctions écrites en python



Noeud Python

Function name permet d'identifier la fonction à exécuter, module path spécifie le chemin du module Python. Le module contient la fonction Python à appeler, session in spécifie une référence à la session Python, session out renvoie une référence à la session Python, return type spécifie le type de données de valeur renvoyée, return valeur est la valeur renvoyée par la fonction de bibliothèque.

- Fermeture de la session python courante



Fermer une session Python

On utilise cette fonction à la fin du programme pour fermer la session python.<sup>[19]</sup>

Il convient de noter que la version Python que l'on choisit ne peut être que 2.7.X ou 3.6.X uniquement. De plus les éditions de Python et LabVIEW doivent être cohérentes, soit on utilise 32 bits en même temps, soit 64 bits.

Les codes python que nous avons utilisés sont des fonctions qui retournent soit un string

(dans le cas où nous avons testé les interfaces Labview/python), soit un tableau de nombres qui représente l'intensité en chaque pixel de l'image (cf. Figure 10), qui est en langage python un `numpy.ndarray`. Une variable de type « `numpy.ndarray` » n'est pas reconnue par Labview : il faut que les fonctions python renvoient des listes. Par exemple, un tableau que l'on appellerait 'noisy\_picture' est une variable (cf. **Annexe 1&2**) de type « `numpy.ndarray` », qui représente un tableau 2D d'intensité. Ce tableau est calculé en prenant le signal émis par chaque particule, auquel on a ajouté un bruit de même statistique que le bruit observé sur des images expérimentales. L'utilisation de LabVIEW ne permet pas l'utilisation directe de la variable de type « `numpy.ndarray` ». Il faut donc convertir 'numpy.ndarray' en 'list' en utilisant `return noisy_picture.tolist()` dans la fonction python.

Au cours de la programmation LabVIEW : il faut indiquer la version spécifique de Python (2.7 ou 3.6) dans **Open Python Session**, il faut aussi indiquer le chemin du module, le nom de la fonction que l'on a déjà créée dans le programme Python, le type de retour (si la fonction Python ne renvoie aucune valeur, ne câblez pas type de retour), les paramètres d'entrée de la fonction Python. Si tout a été bien réglé, on obtiendra la valeur renvoyée par la fonction de bibliothèque à la sortie de **Python Node**.<sup>[20]</sup>

Finalement, on ferme la session Python en utilisant **Close Python Session**.

Les exemples de tests que nous avons utilisés sont reportés en annexes :

- Une fonction test d'affichage d'une variable de type string 'hello world' (cf. **ANNEXE 3**)
- Deux fonctions d'affichage de variables de type `numpy.ndarray` pour simuler des signaux expérimentaux (cf. **ANNEXE 4**)

## 2.3. Affichage Labview d'une image de microscopie obtenue grâce à un programme python

### 2.3.1. Fonctions de base

Les programmes python créés pour faire l'interface python-Labview (*main1.py* et *main2.py* cf. ANNEXE4) nous envoient une valeur 'noisy\_picture.tolist()' de type 'list'. Maintenant, on veut représenter cette valeur de type 'list' en python en LabVIEW. On appelle 'Intensity Graph' ces représentations. Dans les deux exemples, on représente les listes qui sont les valeurs renvoyée de *main1.py* et *main2.py* dans 'Intensity Graph1' et 'Intensity Graph2' :

Si on suit bien les étapes, on peut obtenir une image qui n'est pas exactement celle qu'on attend : cette image et l'image idéale sont symétriques par rapport à l'axe y.

Afin de bien utiliser la valeur renvoyée (`return noisy_picture.tolist()`) qui sortent de **Python Node**, il faut ajouter 'Transpose 2D Array' avant d'écrire les indicateurs 'Intensity Graph1' et 'Intensity Graph2' qui permettent de représenter les deux images sur l'interface.

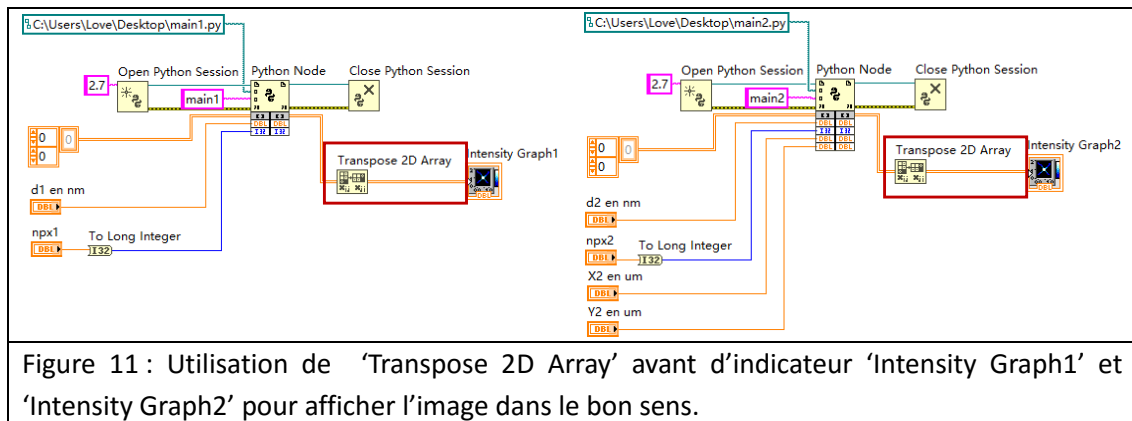


Figure 11 : Utilisation de 'Transpose 2D Array' avant d'indicateur 'Intensity Graph1' et 'Intensity Graph2' pour afficher l'image dans le bon sens.

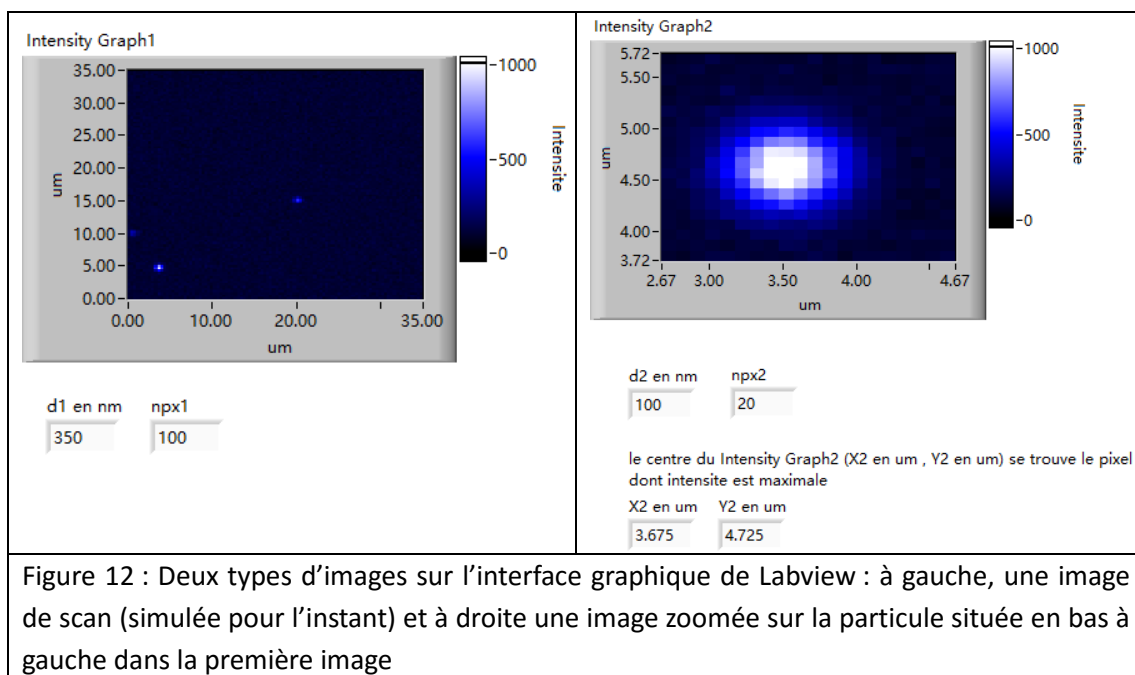
Les images ne sont toutefois pas encore affichées avec la bonne unité. Il faut aussi faire attention dans les propriétés de l'objet à ce que la case 'Autoscale' ne soit pas cochée sur les axes X ou Y du graphique.

Pour l'affichage des images scannées, il a fallu apporter une attention particulière aux échelles des axes et aux coordonnées des pixels. Cela s'est fait en utilisant 'Noeud de propriété' : pour corriger l'unité des axes, on modifie les paramètres 'Scale.Multiplier' et 'Scale.Maximum'

### 2.3.2. Prise en compte d'une fonctionnalité de zoom

Maintenant, on voudrait pouvoir cliquer sur un pixel de l'image 1 et afficher un zoom dans l'image 2. Pour visualiser la position de la souris, il faut d'abord ajouter des curseurs du mode libre (il existe trois modes : Libre, Tracé unique et Tracés multiples) : un selon l'axe x, l'autre selon l'axe y. On souhaite que le point d'intersection des deux curseurs se déplace avec la souris donc on utilise une structure événement (un événement Déplacement de la souris) pour établir le lien entre le curseur et la souris, puis lorsqu'on clique, de récupérer les coordonnées du point en utilisant un autre événement (un événement Souris appuyée/Souris relâchée).

Après avoir fait ces modifications, on obtient deux types d'images (simulées dans cette section) sur l'interface graphique de Labview : une image de scan où on choisit un point autour duquel on va faire le zoom et une image zoomée.

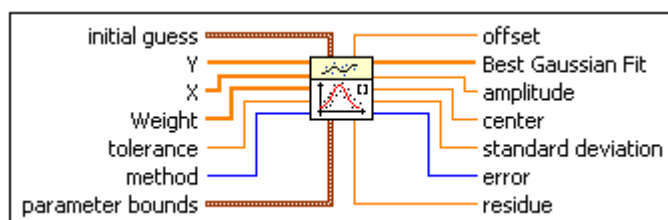


## 2.4. Mise en application : détermination de la position précise des nanoparticules

### 2.4.1. Ajustements gaussiens

L'image d'un objet ponctuel (comme peut être considérée la nanoparticule) par le système de microscopie est une tache lumineuse dont l'intensité est idéalement décrite par une fonction d'Airy. Cette « réponse » du système est dans la réalité suffisamment proche d'une répartition gaussienne pour que l'on puisse considérer qu'un ajustement gaussien représente très bien les taches lumineuses obtenues.

Comme il n'y pas de VI existant en LabVIEW pour faire un ajustement 2D, il nous faut faire deux ajustements 1D en utilisant 'ajustement de pic gaussien (VI)', l'un selon X, l'autre suivant Y. La fonction d'ajustement est la suivante :



Ajustement de pic gaussien (VI)

Renvoie l'ajustement gaussien d'un ensemble de données (**X**, **Y**) en utilisant la méthode des moindres carrés, des moindres résidus absolus, ou la méthode bicarrée. <sup>[21]</sup>

Plus de détails sont donnés en **ANNEXE 5**

Différents modes d'ajustements sont proposés dans le paramètre « method » (Moindres carrés, Moindres résidus absolus, Bicarrée) qui donnent des résultats très similaires. Nous avons choisi de garder la méthode par défaut 'moindres carrés'.

### 2.4.2. Précision de localisation

Nous allons appliquer le code développé sur l'exemple du calcul de l'incertitude de localisation. Cette incertitude dépend en particulier de l'intensité émise par la particule.

#### 2.4.2.1 Calculer l'incertitude de localisation :

Les coordonnées de position  $(x_p, y_p, z_p)$  d'un émetteur fluorescent isolé peuvent être estimées à partir d'une image de microscopie avec une certitude limitée qui comprend à la fois une précision et une exactitude de position (Fig. 13).

Si la coordonnée de position vraie  $x_p$  d'une particule est mesurée plusieurs fois, la précision de localisation décrit la répartition de ces estimations  $x_{p,i}$  autour de sa valeur moyenne  $x_p$ , tandis que l'exactitude de l'emplacement décrit l'écart de la moyenne des coordonnées de position mesurées  $\bar{x}_p$  à partir de la coordonnée de position vraie  $x_p$ . On ne peut pas avoir accès à cette exactitude, mais il est possible de calculer la précision de localisation, communément exprimée en termes d'écart type  $\sigma_x$  :

$$\sigma_x = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_{p,i} - \bar{x}_p)^2}$$

où  $n$  est le nombre d'estimations et les expressions sont similaires pour les coordonnées de position  $y_p$  et  $z_p$ .

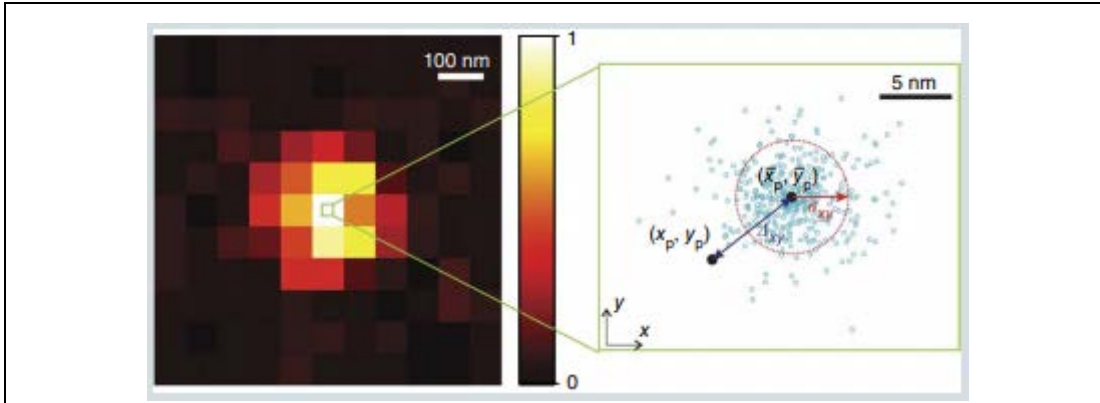
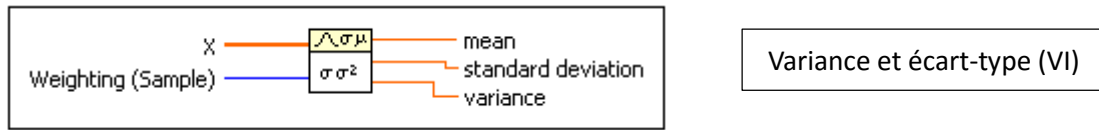


Figure 13 : Visualisation des notions de précision de localisation et d'exactitude de l'emplacement, tirée de [ref 22] <sup>[22]</sup>. Un exemple de l'image enregistrée expérimentalement d'un seul émetteur est montrée. La position réelle des particules  $(x_p, y_p)$  peut être estimée à partir d'une telle image avec une précision de localisation latérale (lateral localization precision)  $\sigma_{xy} = (\sigma_x + \sigma_y)^{1/2}$  et une exactitude de l'emplacement latérale (lateral localization accuracy)  $\Delta_{xy} = (\Delta_x + \Delta_y)^{1/2}$ . Les cercles bleus indiquent estimations des positions déterminées expérimentalement à partir de différentes images du même émetteur, et  $(\bar{x}_p, \bar{y}_p)$  est la moyenne de ces valeurs individuelles.

LabVIEW permet de faire ces calculs :

- D'abord, pour calculer la précision de localisation selon x et y ( $\sigma_x$  et  $\sigma_y$ ), on utilise 'Variance et écart-type (VI)', puis choisit le mode 'Sample standard deviation'.

[23]



- Dès que l'on obtient  $\sigma_x$  et  $\sigma_y$ , on calcule la précision de localisation latérale (lateral localization precision)  $\sigma_{xy} = \sqrt{(\sigma_x + \sigma_y)}$

#### 2.4.2.2 Tracer la courbe d'incertitude de localisation en fonction de l'intensité

L'idée est de programmer LabVIEW pour qu'il estime la position d'une particule un grand nombre de fois. La particule est à la même position et on la localise plusieurs fois. La position estimée par le programme n'est pas toujours la même à cause du bruit qui est aléatoire (comme dans l'expérience) et finalement une distribution de points correspondra à une particule. Plus cette distribution est étalée est plus notre incertitude sur la position sera élevée.

On utilise *main.py* (cf. ANNEXE 6) pour simuler de cette expérience. On produit un scan représentant une unique particule dont on fait artificiellement diminuer l'intensité, d'une valeur maximale de 6000 (unités arbitraires à zéro par pas de 50. Pour une valeur donnée, on construit une statistique de localisation avec 40 itérations de l'expérience. A chaque essai on obtient  $(x_p, y_p)$  en utilisant 'ajustement de pic gaussien (VI)', puis on calcule la précision de localisation latérale pour chaque intensité donnée. On obtient ainsi la courbe représentée Figure 14. On observe bien que l'incertitude diminue lorsque l'intensité augmente. La loi suivie est théoriquement en  $1/\sqrt{I}$ .

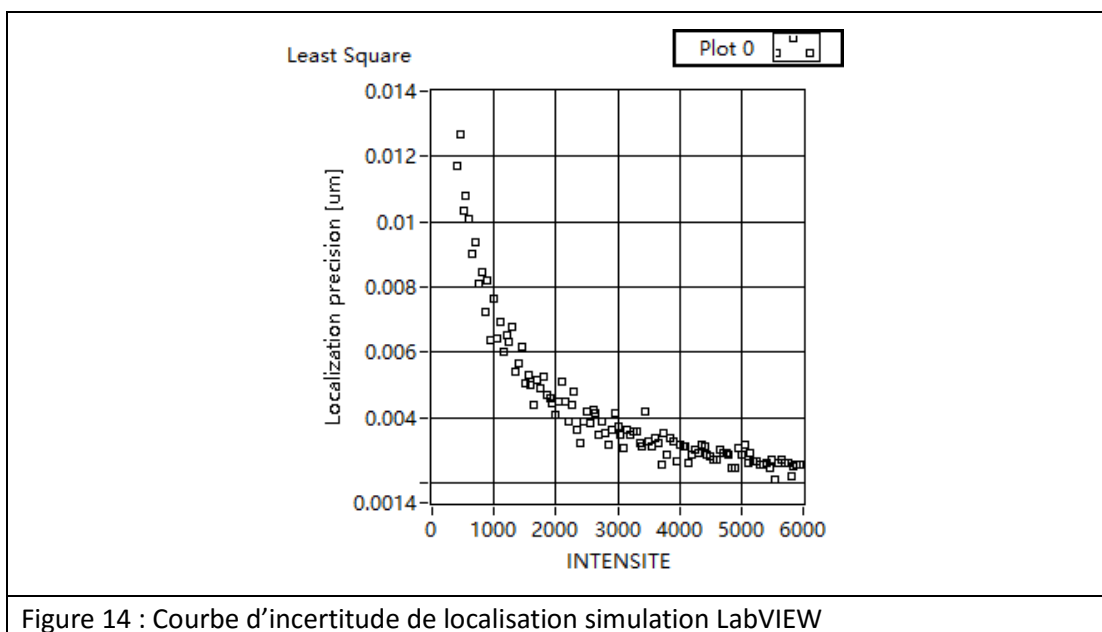


Figure 14 : Courbe d'incertitude de localisation simulation LabVIEW

### 3. Expérience

Lorsque j'ai pu venir au laboratoire, après la période de confinement, nous avons pu travailler directement sur l'expérience. Afin de tester en condition réelle les codes sur lesquels nous avons travaillé, nous avons utilisé des billes fluorescentes placées sur une lamelle de verre. Ces billes, de taille caractéristique 1 micromètre, émettent de la lumière autour d'une longueur d'onde de 660nm lorsqu'elles sont éclairées par un laser à 633nm.

#### 3.1. Piloter la platine piézoélectrique

##### 3.1.1 Principe du scan piézo électrique :

Nous devons prendre en compte le pilotage de la platine de translation piézoélectrique par le logiciel LabVIEW. Il faut donc établir une liaison entre notre programme et le contrôleur 3 axes à l'aide en passant par une connexion USB. Le principe est de changer la tension qui sort du contrôleur pour modifier les positions des piézoélectriques en utilisant l'interface USB.

On obtient la première image en déplaçant la platine piézo-électrique : il s'agit donc d'une méthode mécanique. Le temps de déplacement dépend de la masse à déplacer. Dans notre expérience ce temps de déplacement d'un point de départ à un point d'arrivée est d'environ 10 ms.

##### 3.1.2 Code LabVIEW à réaliser

###### 3.1.2.1 Commande du piézo-électrique

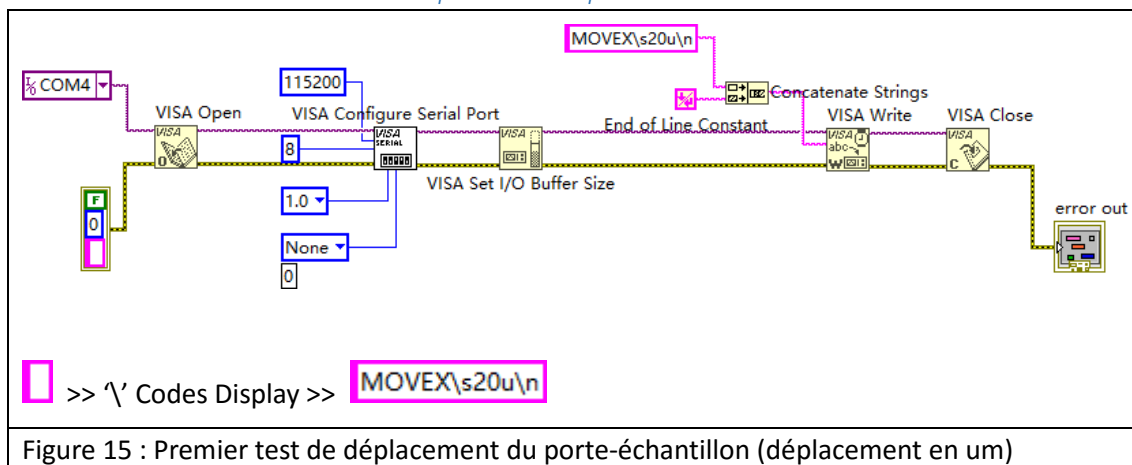


Figure 15 : Premier test de déplacement du porte-échantillon (déplacement en um)

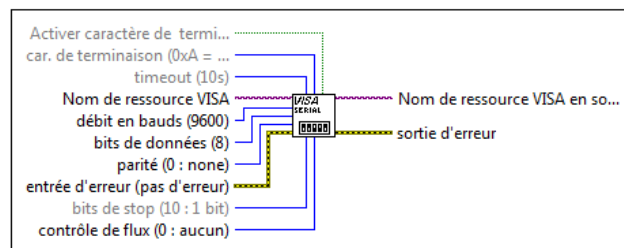
Premièrement, on utilise la fonction '**VISA Open (fonction)**' à ouvrir une session sur le périphérique spécifié par Nom de ressource VISA et renvoyer un identificateur de session pouvant servir à appeler n'importe quelles autres opérations de ce périphérique. <sup>[24]</sup>

D'après le document 'Liste des commandes pour l'interface USB' (cf. **ANNEXE 7**), on obtient les paramètres pour le port de communication :

**Baud rate : 115200 Bauds      Data bits : 8      Stop bits : 1**  
**No parity Flow control : none Character for transmission end : LF (0xA)**



Les paramètres ci-dessus nous permettent de paramétrer l'interface USB en utilisant '**VISA Configure Serial Port (Instr).vi**'.<sup>[25]</sup>



VISA Configure Serial Port

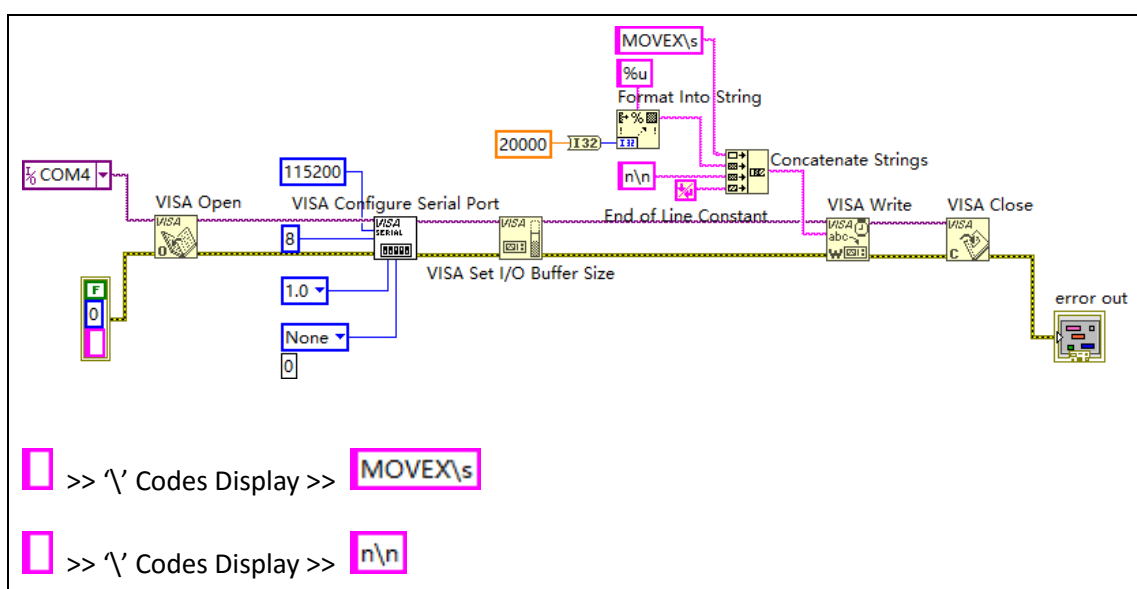
Initialise le port série spécifié par **Nom de ressource VISA** avec les paramètres spécifiés. Câblez des données à l'entrée **Nom de ressource VISA** pour déterminer l'instance polymorphe à utiliser ou sélectionnez.

Après ce vi, j'ajoute '**VISA Set I/O Buffer Size (Function)**' pour définir la taille du buffer à 4096 octets (valeur par convention). (il faut définir la taille comme étant légèrement plus grande que la quantité de données que l'on prévoit de transmettre ou de recevoir).

Pour changer la position de la platine de translation piezo, il faut entrer les commandes de 5 caractères dans la fonction '**VISA WRITE (fonction)**' (cf. **ANNEXE 7**)

Finalement, on utilise '**VISA Close (fonction)**' pour fermer la session sur le périphérique ou l'objet événement spécifié par Nom de ressource VISA.<sup>[26]</sup>

Dans le cas de notre expérience, on voudrait définir la valeur du déplacement comme une variable en nanomètres. Donc on a besoin de la fonction 'Concatenate Strings (fonction)' à concaténer les chaînes et les tableaux de chaînes unidimensionnelles en entrée en une seule chaîne en sortie.<sup>[27]</sup>





 >> Normal Display >>  %u

Figure 16 : Deuxième test de déplacer le porte-échantillon qui ressemble à notre cas de l'expérience (déplacement en nm)

#### 3.1.2.2 code LabVIEW pour faire le scan piézo électrique

La structure que j'ai écrite pour faire le scan piézo électrique est constituée de deux structures : la première structure déplace l'échantillon selon la direction y, l'autre le déplace selon la direction x.

D'abord, à y fixé, on se déplace le long de l'axe x avec un pas d1 (la taille d'un pixel en nanomètre) ensuite on mesure l'intensité correspondante, le résultat de ce scan est un tableau 1D. Puis on se déplace le long de l'axe y d'une distance d1 aussi, on fait le scan selon x comme avant, on combine ce nouveau tableau 1D avec l'ancien. On obtient ainsi un tableau 2D. On rafraichit ce tableau 2D quand on déplace l'axe y, de sorte que l'échelle d'intensité s'adapte de manière dynamique. Finalement, on obtient un scan piézo électrique quand l'axe y déplace jusqu'au bout.

Quand on fait le scan piézo électrique, on souhaite que le laser soit focalisé au foyer image de l'objectif. Le faisceau laser passant par le DMD avant d'arriver au microscope, il faut utiliser un hologramme de référence qui envoie la lumière du laser dans la direction de l'axe optique du microscope. On crée pour cela deux fonctions python **displayref** et **shutdownref** pour piloter le DMD :

- Avant de faire le scan, on exécute une fois la fonction **displayref** pour afficher sur le DMD un hologramme de référence qui envoie la lumière selon l'axe optique de l'objectif.
- Après le scan, on exécute la fonction **shutdownref** pour mettre le DMD en état inactif.

#### 3.2. Obtention de la première image

L'échantillon sur lequel nous avons testé notre système est une lamelle de verre sur laquelle ont été déposées des billes fluorescentes. Ces billes ont un diamètre de l'ordre du micromètre et émettent de la lumière dans le rouge à 670nm lorsqu'elles sont éclairées à 633nm. La figure 17 montre à droite l'image obtenue par un scan piézoélectrique de cet échantillon.

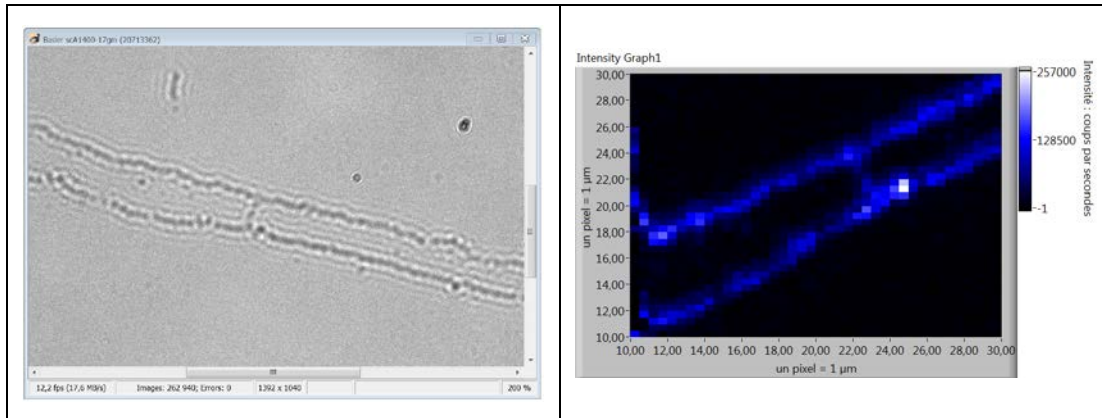


Figure 17 : A gauche, l'image en lumière blanche obtenue pour des billes fluorescentes. On peut voir qu'elles sont disposées en ligne suite à la manière dont elles ont été déposées sur la lamelle de verre. A droite, l'image obtenue par balayage à l'aide du programme Labview que j'ai réalisé. L'image est inversée par rapport à celle en lumière blanche mais on distingue clairement la structure spécifique de la disposition des particules. Le programme permet de cliquer sur un pixel pour zoomer ensuite (cf. section suivante)

### 3.3. Obtention de la deuxième image (zoom)

#### 3.3.1 Le principe du scan en holographie numérique

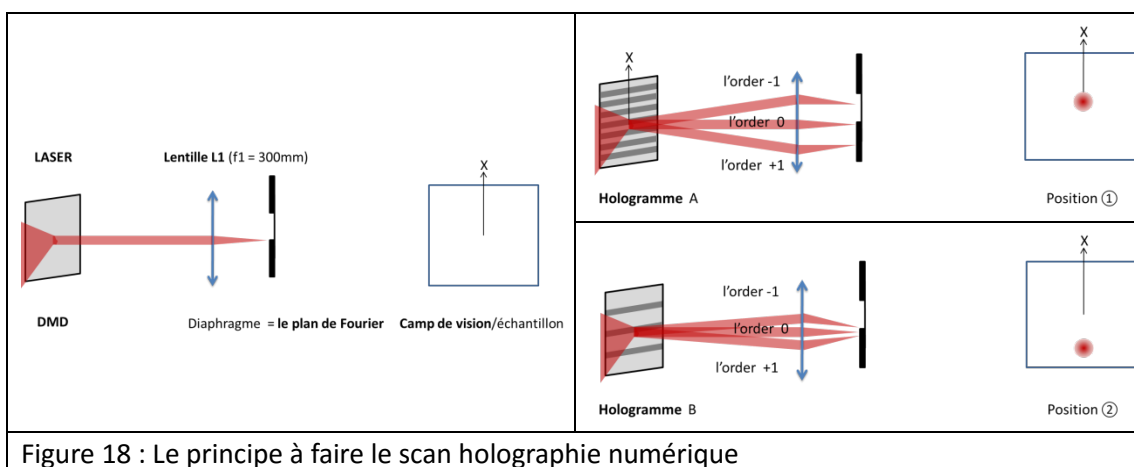


Figure 18 : Le principe à faire le scan holographie numérique

Maintenant, on ne déplace plus le porte-échantillon, comme c'était le cas pour l'acquisition de la première image, mais on déplace le spot laser lui-même en utilisant le DMD. Avant de commencer, on utilise Python pour piloter le DMD et générer une bibliothèque d'hologrammes, chaque hologramme correspond à une position différente sur l'échantillon (①, ② ...). Pour chaque hologramme, on utilise le programme LabVIEW pour compter les photons (intensité) (ce programme est le même que on a utilisé à compter les photons dans notre programme qui fait le scan piézoélectrique).

La fréquence de modification de l'hologramme est élevée (plusieurs dizaines de kHz) ce qui permet de réaliser plus rapidement un scan avec autant de pixels. Ce deuxième scan est mieux résolu mais sur une zone plus petite. Donc on obtient l'image 2 avec une méthode

d'optique interférence (holographie), contrairement à la méthode mécanique, cette méthode est indépendante de la masse de l'échantillon et pendant notre expérience, comme ce n'est pas un déplacement mécanique, le temps entre deux points est aussi indépendant de la distance qui sépare ces points.

### 3.3.2 Code LabVIEW à réaliser

Une fois que l'on a obtenu le premier scan piézo électrique, comme dans la partie simulation :

- On clique sur un pixel du premier scan
- On déduit les coordonnées du point autour duquel on veut zoomer. Les coordonnées du pixel dont l'intensité est maximale servent à définir le centre du deuxième graphe.
- On exécute un événement pour déplacer le 'trois axe contrôleur' à la position où l'intensité est maximale en utilisant les coordonnées précédentes
- On clique sur un bouton « bibgen » à créer les bibliothèques d'une séquence d'hologramme

(On envoie les paramètres en entrée d'une fonction python)

- On clique le bouton « SCAN2 » qui est lié avec une fonction « **scan** » créée en Python. La platine piézoélectrique reste immobile mais la séquence d'hologramme est affichée et le laser scanne autour du pixel d'intensité maximale dans le scan1
- On affiche le résultat du nouveau scan : On acquiert ainsi la deuxième image avec les paramètres d2 (la taille d'un pixel en nanomètre) ET npx2 (le nombre de pixels)..

### 3.3.3 Obtention de la deuxième image

La figure 19 est obtenue par le scan holographie numérique après avoir cliqué sur une position (le blanc) de la figure 17 où l'intensité est plus intense.

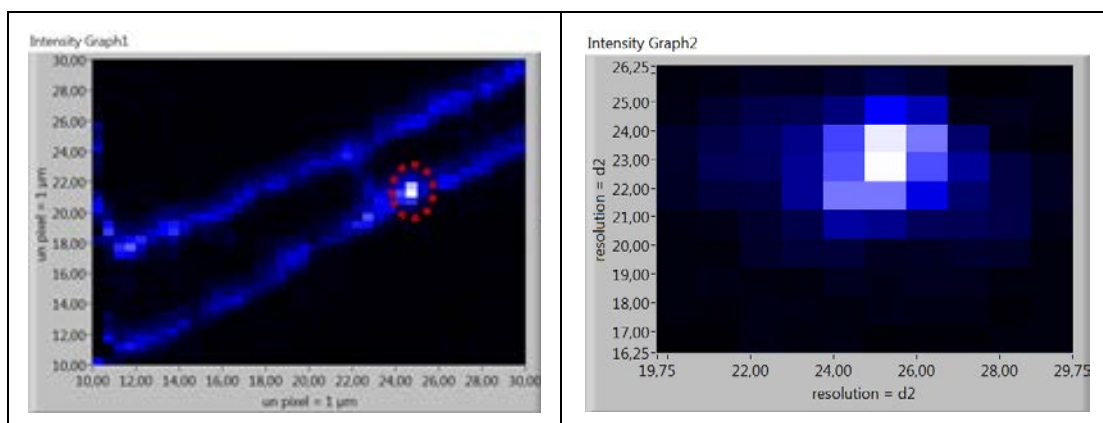


Figure 19 : Scan holographie numérique : Après avoir fait un clic sur le scan piézo électrique (figure de gauche), on fait un scan plus précis et plus rapidement avec le DMD qui déplace le spot laser (figure de droite). L'image obtenue est une image zoomée sur la partie entourée de pointillés rouges dans la figure de gauche.

## 4. Conclusion

Bilan de l'avancée du système entre le moment où je suis arrivé et le moment où je pars :

Avant mon arrivée, Florian avait déjà monter une grande partie de l'expérience. Il avait codé

des fonctions Python : **displayref** et **shutdownref** pour piloter le DMD, une fonction **bibgen** pour créer les bibliothèques de séquence d'hologrammes et une fonction **scan** pour passer d'un hologramme (créé par la fonction **bibgen**) à l'autre. Il avait aussi codé un programme Labview pour le comptage de photons.

Mon travail a été de faire l'interface python-labview (c'est-à-dire l'exécution de tous les codes écrits en python et labview dans un programme principal écrit en Labview) et aider Florian à faire l'alignement optique de la voie de collection.

Pendant le confinement (télétravail 11/05/2020 – 22/06/2020), j'ai écrit trois codes importants pour simuler les expériences : le premier code qui fait deux scans et l'ajustement gaussien ; lorsqu'on a les coordonnées  $(x_p, y_p)$  obtenues par l'ajustement, j'ai écrit le deuxième code pour tracer la courbe d'incertitude de localisation ; le troisième code est utilisé pour comparer trois méthodes d'ajustement.

Pendant la période de 22/06/2020 à 31/07/2020, je suis venu au labo, j'ai ainsi pu modifier le premier code de la simulation et l'adapter au cas expérimental : d'abord en ajoutant une structure dans notre programme labview pour piloter la platine piézoélectrique, puis en adaptant les fonctions écrites en python par Florian à la structure de comptage photons dans notre programme principal. Finalement on a réconstitué les structures dans un programme global et remplacé certaines parties par des sous-vi.

J'ai ainsi pu réaliser deux scans en utilisant mon programme labview, un scan « grossier » à l'aide du piezoélectrique, l'autre scan « précis » à l'aide des hologrammes. Les coordonnées précises de la particule  $(x_p, y_p)$  sont ensuite obtenues en faisant l'ajustement gaussien sur la dernière image.

Ce travail va permettre à Florian de commencer à étudier des systèmes dynamiques avec des particules en mouvement. Dans ce cas de figure, les hologrammes seront utilisés pour localiser la particule en direct, et maintiendront le spot laser sur la particule plutôt que de faire un scan global. Le dispositif devrait permettre de mesurer des trajectoires en localisant à tout instant l'émetteur à mieux que 20nm et avec une résolution temporelle de l'ordre de la milliseconde.

#### **Bilan personnel :**

D'un point de vue personnel, ce stage m'a permis d'enrichir mes compétences en labview : pendant mes études à l'IUT, j'ai codé des programmes en TD/TP ou en projet, mais dans un cadre bien balisé et défini. Cette fois, j'ai vu des problèmes comme interface python-labview, l'ajustement de pic gaussien qui ne nous avaient pas été présentés en cours, donc j'ai dû trouver des solutions, ce qui prend un temps significatif. Ce stage m'a aussi aidé à appliquer beaucoup de connaissances acquises : par exemple en TP, nous avons utilisé une interface USB pour piloter une caméra, pendant mon stage, j'ai rencontré un nouvel appareil 'la platine piézoélectrique', qui étaient pilotée de la même manière.

Le travail du stage s'est réalisé dans un mode projet. J'ai pu voir que le travail en équipe peut être très efficace, chacun s'occupe de sa partie, en concertation avec les autres, et les résultats peuvent arriver rapidement au final.

## 5. Références

- [1] <http://www.lumin.universite-paris-saclay.fr/fr>
- [2] [\*Geng, Q., Gu, C., Cheng, J. & Chen, S. Digital micromirror device-based two-photon microscopy for three-dimensional and random-access imaging. Optica 4, 674 \(2017\).\*](#)
- [3] [\*Cheng, J., Gu, C., Zhang, D., Wang, D. & Chen, S.-C. Ultrafast axial scanning for two-photon microscopy via a digital micromirror device and binary holography. Opt. Lett. 41, 1451 \(2016\).\*](#)
- [4] [\*Semmer, F., Introduction à Kaiyuan de notre projet.pdf, p1\*](#)
- [5] [\*Semmer, F., Introduction à kaiyuan de notre projet.pdf, p3\*](#)
- [6] [\*Semmer, F., DEVELOPPEMENT D'UN DISPOSITIF DE BALAYAGE DE FAISCEAU, EN TROIS DIMENSIONS, PAR HOLOGRAPHIE EN VUE DE SON INTEGRATION DANS UN MICROSCOPE POUR LE SUIVI DE NANOPARTICULES INDIVIDUELLES.pdf, p4\*](#)
- [7] [\*Semmer, F., SchemaConclusionFe.pptx\*](#)
- [8] <https://www.ti.com/graphics/folders/partimages/DLP7000.jpg>
- [9] [\*Semmer, F., DEVELOPPEMENT D'UN DISPOSITIF DE BALAYAGE DE FAISCEAU, EN TROIS DIMENSIONS, PAR HOLOGRAPHIE EN VUE DE SON INTEGRATION DANS UN MICROSCOPE POUR LE SUIVI DE NANOPARTICULES INDIVIDUELLES, p5-7\*](#)
- [10] <https://fr.wikipedia.org/wiki/Pi%C3%A9zo%C3%A9lectricit%C3%A9>
- [11] <http://www.piezoconcept.com/products-3-axis-stages/lt3/>
- [12] <http://www.piezoconcept.com/wp-content/uploads/2017/03/ANALOGUE-CONTROLLER-0to10V-4Amps.pdf>
- [13] <http://www.piezoconcept.com/wp-content/uploads/2013/10/LT-3-ALU.png>
- [14] <http://www.piezoconcept.com/wp-content/uploads/2013/02/3-axis-Controller-PRIO.png>
- [15] [https://en.wikipedia.org/wiki/Photon\\_counting](https://en.wikipedia.org/wiki/Photon_counting)
- [16] [https://www.excelitas.com/product/spcm-aqrh-trPD\\_SPCM\\_Family\\_brochure\\_June2019.pdf](https://www.excelitas.com/product/spcm-aqrh-trPD_SPCM_Family_brochure_June2019.pdf)
- [17] <https://www.ni.com/fr-fr/innovations/white-papers/08/fpga-fundamentals.html>
- [18] [https://www.microscope.healthcare.nikon.com/fr\\_EU/products/inverted-microscopes/eclipse-ti-series](https://www.microscope.healthcare.nikon.com/fr_EU/products/inverted-microscopes/eclipse-ti-series)
- [19] [http://zone.ni.com/reference/fr-XX/help/371361R-0114/glang/python\\_pal/](http://zone.ni.com/reference/fr-XX/help/371361R-0114/glang/python_pal/)
- [20] [http://zone.ni.com/reference/fr-XX/help/371361R-0114/glang/python\\_node/](http://zone.ni.com/reference/fr-XX/help/371361R-0114/glang/python_node/)
- [21] [http://zone.ni.com/reference/fr-XX/help/371361R-0114/gmath/gaussian\\_peak\\_fit/](http://zone.ni.com/reference/fr-XX/help/371361R-0114/gmath/gaussian_peak_fit/)
- [22] <https://www.nature.com/articles/nmeth.2843>
- [23] [https://zone.ni.com/reference/fr-XX/help/371361R-0114/gmath/stand\\_deviation\\_variance/](https://zone.ni.com/reference/fr-XX/help/371361R-0114/gmath/stand_deviation_variance/)
- [24] [http://zone.ni.com/reference/fr-XX/help/371361R-0114/lvinstio/visa\\_open/](http://zone.ni.com/reference/fr-XX/help/371361R-0114/lvinstio/visa_open/)
- [25] [https://zone.ni.com/reference/en-XX/help/371361R-01/lvinstio/visa\\_configure\\_serial\\_port/](https://zone.ni.com/reference/en-XX/help/371361R-01/lvinstio/visa_configure_serial_port/)
- [26] [http://zone.ni.com/reference/fr-XX/help/371361R-0114/lvinstio/visa\\_close/](http://zone.ni.com/reference/fr-XX/help/371361R-0114/lvinstio/visa_close/)
- [27] [http://zone.ni.com/reference/fr-XX/help/371361R-0114/glang/concatenate\\_strings/](http://zone.ni.com/reference/fr-XX/help/371361R-0114/glang/concatenate_strings/)

## 6. Annexes

Dans cette partie : tout ce qui est très technique (pas utile à la compréhension globale de ce que j'ai fait)

### TABLE DES ANNEXES

#### **Annexe 1**    *main1 - python.py*

un code python qui simule le premier scan expérimental 'scan « grossier » piezo'

#### **Annexe 2**    *main2 - python.py*

un code python qui simule le deuxième scan expérimental 'scan « précis » holo'

#### **Annexe 3**    Une variable de type string 'Hello world'

un code python initial qui peut afficher 'Hello world', un code python modifié pour faire l'interface python-labview, un programme Labview pour faire l'interface

#### **Annexe 4**    Une variable de type `numpy.ndarray`

un code python initial qui peut afficher un graphe 2D, un code python modifié pour faire l'interface python-labview, un programme Labview pour faire l'interface

#### **Annexe 5**    Ajustement de pic gaussien

deux tableaux en 1D (une série de coordonnée x, l'autre y) qui représente une série de point de la forme gaussienne, un programme Labview pour faire l'ajustement de pic gaussien

#### **Annexe 6**    *main.py*

un code python comme celui de l'annexe 4 mais on ajoute un paramètre : l'intensité

#### **Annexe 7**    Liste des commandes pour l'interface USB

la documentation aidant à faire l'interface USB

---

## Annexe 1 *main1 - python.py*

---

```
# : 211 0 0
def : 255 119 0
fonc() : 0 0 255
string : 0 170 0
list(None) : 144 0 144
-----
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import matplotlib.pyplot as plt

d1=350. # résolution du scan en nanomètre
npx1=100 # nombre de pixel du scan

picture=np.zeros([npx1,npx1]) # fonction numpy qui initie un array de taille npx12 avec
des zéros
sigma=1000 # largeur caractéristique de la gaussienne
nbspot=3 # nombre de particules dans le champs simulé

tailleimage=12*int(sigma/d1) # sigma/d correspond converti l'écart des gaussiennes en
pixel. tailleimage est le nombre de pixel que l'on utilise
picture=np.zeros([npx1,npx1]) # on réinitialise le tableau

def gauss2D(x,y,I,sigma): ## définir la fonction gaussienne:
    return I*np.exp(-(x**2+y**2)/(2*sigma**2))

In = [300,1000,500] ## créer trois particules : [500,10000,300] [3513,4625,1000]
[20000,15000,500]
xn = [500,3513,20000]
yn = [10000,4625,15000]

In = np.asarray(In) ## convertir 'list' en 'numpy.ndarray'
xn = np.asarray(xn)
yn = np.asarray(yn)
```



```

for i in range(npx1):
    for j in range(npx1):
        picture[i,j] = sum(gauss2D(j*d1-xn,i*d1-yn,ln,sigma)) # CALCUL DE
L'INTENSITÉ EN CHAQUE POINT DE COORDONÉE (x,y) : somme des gaussienne
centrées sur les trois particules de coordonnées (xn,yn)

N = 100 ## ajouter du bruit:
background = np.ones([npx1,npx1])*N ## Background est un niveau de signal de fond,
dont la moyenne est N,
noisy_background = np.random.normal(background,np.sqrt(background)) ## auquel on
rajout du bruit à l'aide d'une loi normale.
noisy_picture = np.random.poisson(picture+noisy_background) ## Puis on ajoute les
particules à ce fond bruté (picture+noisy_background) auquel on ajoute un second bruit
qu'on appelle bruit de Poisson

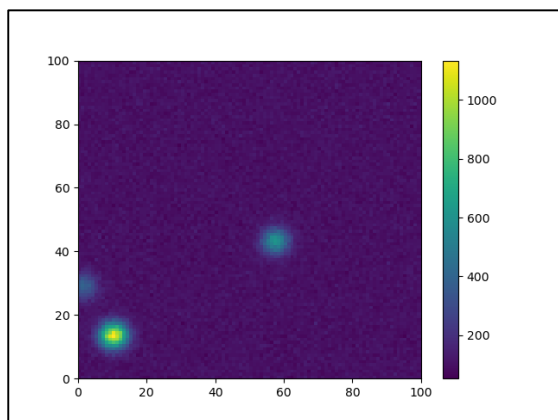
plt.pcolormesh(noisy_picture)
##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
plt.colorbar()

plt.figure()
plt.imshow(noisy_picture)

##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
##plt.colorbar()

plt.show()

```



Scan «grossier» piezo :  
résolution du scan (d1) =350nm |  
nombre de pixel du scan (npx1) = 100

---

## Annexe 2 *main2 - python.py*

---

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import matplotlib.pyplot as plt

d2=100 #résolution du scan en nanomètre
npx2=20 # nombre de pixel du scan

c = [3.675,4.725] #coordonnées du centre de l'image en microns

c = np.asarray(c)*1000
x = np.linspace(c[0]-npx2//2*d2,c[0]+npx2//2*d2,npx2)
y = np.linspace(c[1]-npx2//2*d2,c[1]+npx2//2*d2,npx2)

picture=np.zeros([npx2,npx2]) # fonction numpy qui initie un array de taille npx22 avec
des zéros
sigma=300 # largeur caractéristique de la gaussienne
nbspot=3 # nombre de particules dans le champs simulé

tailleimage=12*int(sigma/d2) # sigma/d correspond converti l'écart des gaussiennes en
pixel. tailleimage est le nombre de pixel que l'on utilise pour simuler une mage
picture=np.zeros([npx2,npx2]) # on réinitialise le tableau

def gauss2D(x,y,I,sigma): ## définir la fonction gaussienne:
    return I*np.exp(-(x**2+y**2)/(2*sigma**2))

In = [300,1000,500] ## créer trois particules : [500,10000,300] [3513,4625,1000]
[20000,15000,500]
xn = [500,3513,20000]
yn = [10000,4625,15000]

In = np.asarray(In) ## convertir 'list' en 'numpy.ndarray'
xn = np.asarray(xn)
yn = np.asarray(yn)
```

```
for i in range(npx2):
    for j in range(npx2):
        picture[i,j] = sum(gauss2D(x[j]-xn,y[i]-yn,In,sigma)) # CALCUL DE
L'INTENSITÉ EN CHAQUE POINT DE COORDONÉE (x,y) : somme des gaussienne
centrées sur les trois particules de coordonnées (xn,yn)
```

N = 100 ## ajouter du bruit:

background = np.ones([npx2,npx2])\*N ## Background est un niveau de signal de fond,  
dont la moyenne est N,

noisy\_background = np.random.normal(background,np.sqrt(background)) ## auquel on  
rajout du bruit à l'aide d'une loi normale.

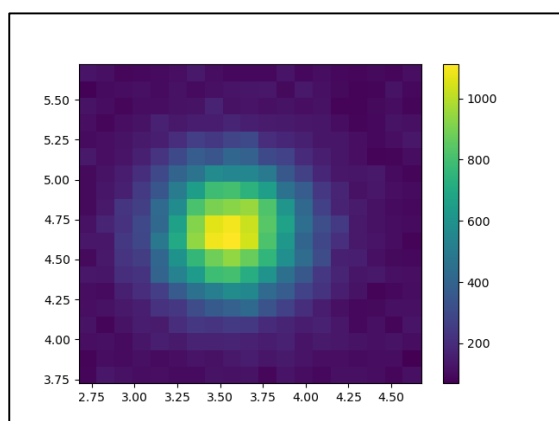
noisy\_picture = np.random.poisson(picture+noisy\_background) ## Puis on ajoute les  
particules à ce fond bruté (picture+noisy\_background) auquel on ajoute un second bruit  
qu'on appelle bruit de Poisson

```
plt.pcolormesh(x/1000,y/1000,noisy_picture)
##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
plt.colorbar()
```

```
#plt.figure()
#plt.imshow(noisy_picture)
```

```
##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
##plt.colorbar()
```

```
plt.show()
```



Scan «précis» holo :  
résolution du scan (d2) =100nm |  
nombre de pixel du scan (npx2) = 20

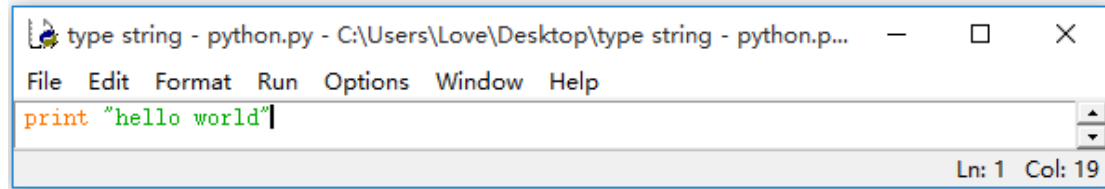
---

## Annexe 3 Une variable de type string 'hello world'

---

`main()`

/code Python/

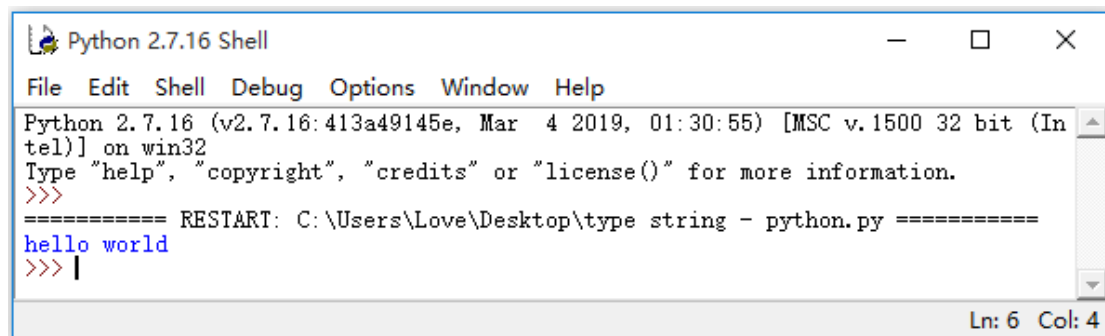


type string - python.py - C:\Users\Love\Desktop\type string - python.p... — □ ×

File Edit Format Run Options Window Help

```
print "hello world"
```

Ln: 1 Col: 19



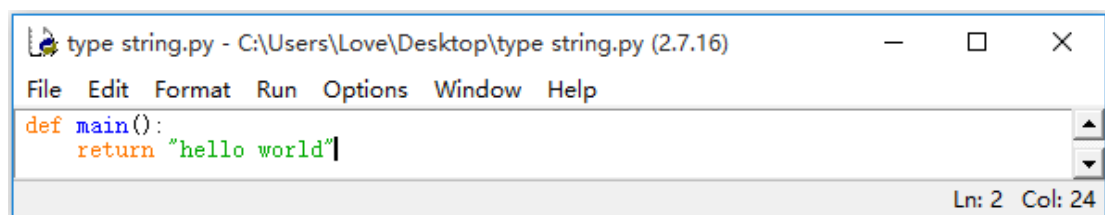
Python 2.7.16 Shell — □ ×

File Edit Shell Debug Options Window Help

Python 2.7.16 (v2.7.16:413a49145e, Mar 4 2019, 01:30:55) [MSC v.1500 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:\Users\Love\Desktop\type string - python.py =====  
hello world  
>>> |

Ln: 6 Col: 4

/ interactions Python et LabVIEW/

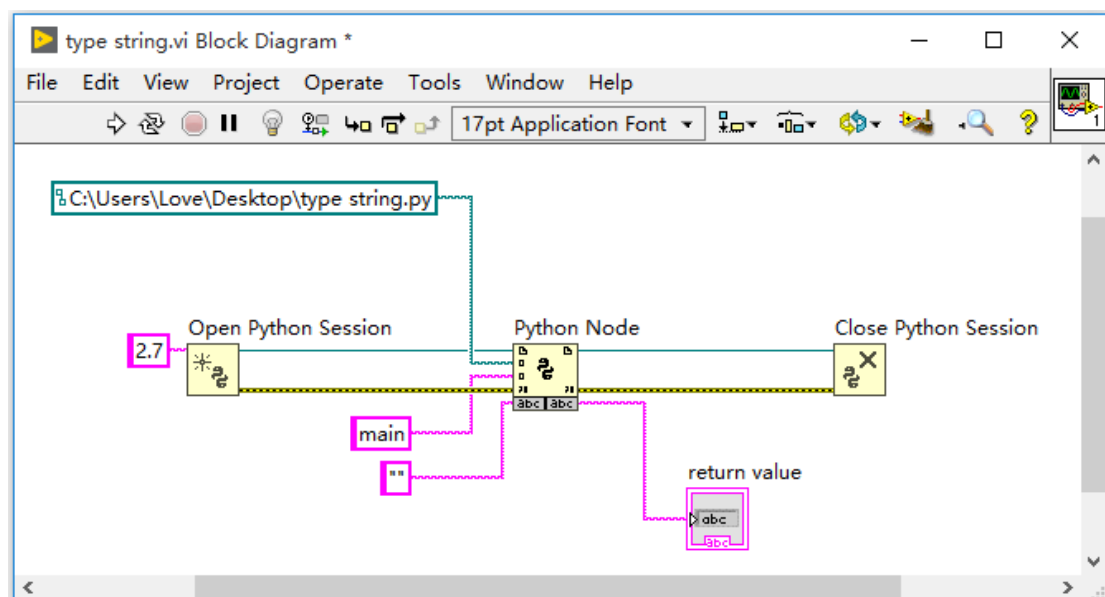


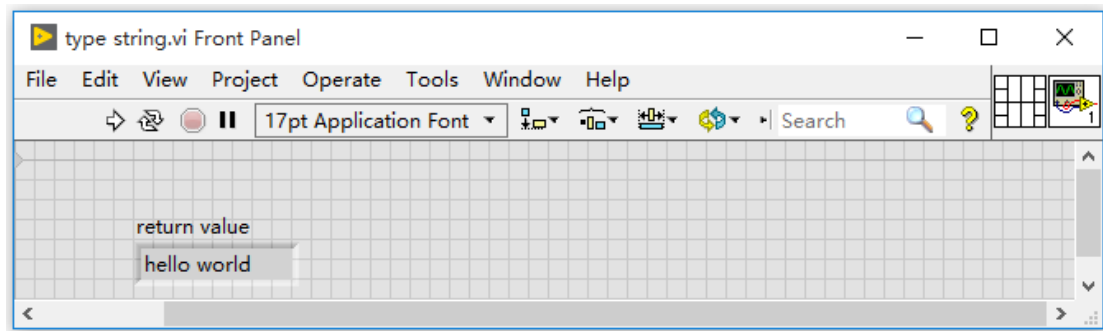
type string.py - C:\Users\Love\Desktop\type string.py (2.7.16) — □ ×

File Edit Format Run Options Window Help

```
def main():  
    return "hello world"
```

Ln: 2 Col: 24






---

## Annexe 4 Une variable de type numpy.ndarray

---

Main1()

/code Python : *main1 - python.py* /

**#!/usr/bin/env on**

**# coding: utf-8**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

d1=350.

np1=100

picture=np.zeros([np1,np1])

sigma=1000

nbspot=3

tailleimage=12\*int(sigma/d1)

picture=np.zeros([np1,np1])

**def** gauss2D(x,y,I,sigma):

**return** I\*np.exp(-(x\*\*2+y\*\*2)/(2\*sigma\*\*2))

In = [300,1000,500]

xn = [500,3513,20000]

yn = [10000,4625,15000]

In = np.asarray(In)

xn = np.asarray(xn)

yn = np.asarray(yn)

```

for i in range(npx1):
    for j in range(npx1):
        picture[i,j] = gauss2D(j*d1-xn,i*d1-yn,In,sigma))

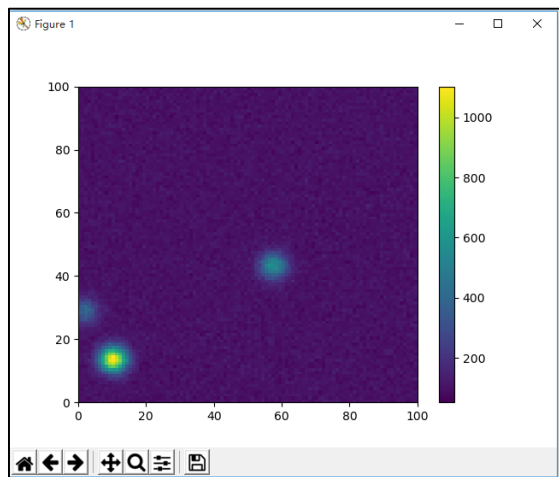
N = 100
background = np.ones([npx1,npx1])*N
noisy_background = np.random.normal(background,np.sqrt(background))
noisy_picture = np.random.poisson(picture+noisy_background)

plt.pcolormesh(noisy_picture)
##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
plt.colorbar()

plt.figure()
plt.imshow(noisy_picture)

##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
##plt.colorbar()
plt.show()

```



Scan «grossier» piezo :  
 resolution du scan (d1) =350nm |  
 nombre de pixel du scan (npx1) = 100

---

/ interactions Python et LabVIEW : *main1.py* /

```

#!/usr/bin/env python
# coding: utf-8

```

```

import numpy as np
import matplotlib.pyplot as plt

```

```
def main1(d1,npx1):
```

```
    picture=np.zeros([npx1,npx1])
```

```
    sigma=300
```

```
    nbspot=3
```

```
    tailleimage=12*int(sigma/d1)
```

```
    picture=np.zeros([npx1,npx1])
```

```
def gauss2D(x,y,I,sigma):
```

```
    return I*np.exp(-(x**2+y**2)/(2*sigma**2))
```

```
In = [300,1000,500]
```

```
xn = [500,3513,20000]
```

```
yn = [10000,4625,15000]
```

```
In = np.asarray(In)
```

```
xn = np.asarray(xn)
```

```
yn = np.asarray(yn)
```

```
for i in range(npx1):
```

```
    for j in range(npx1):
```

```
        picture[i,j] = sum(gauss2D(j*d1-xn,i*d1-yn,In,sigma))
```

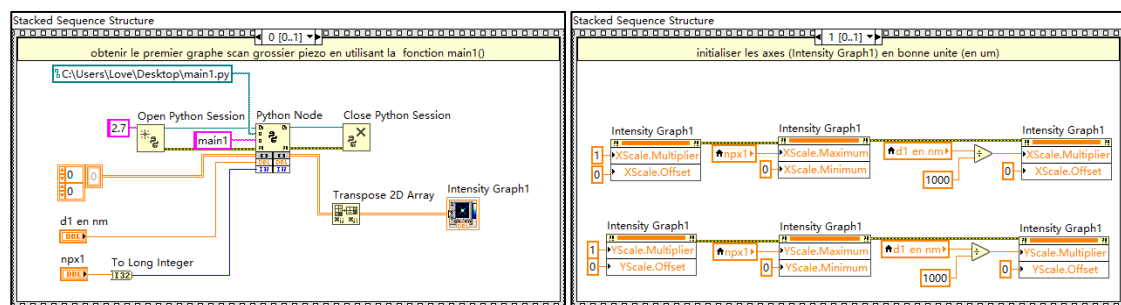
```
N = 100
```

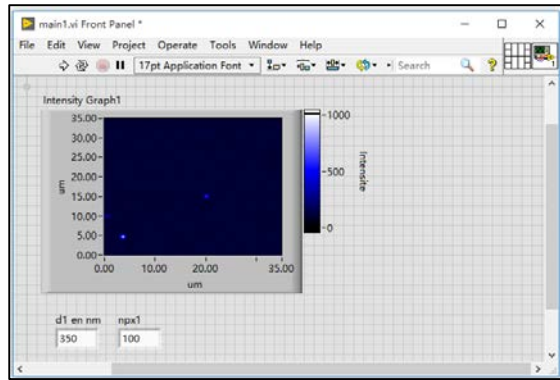
```
background = np.ones([npx1,npx1])*N
```

```
noisy_background = np.random.normal(background,np.sqrt(background))
```

```
noisy_picture = np.random.poisson(picture+noisy_background)
```

```
return noisy_picture.tolist()
```






---

**Main2()**

/code Python : *main2 - python.py* /

**#!/usr/bin/env python**

**# coding: utf-8**

**import numpy as np**

**import matplotlib.pyplot as plt**

**d2=100**

**npx2=20**

**c = [3.675,4.725]**

**c = np.asarray(c)\*1000**

**x = np.linspace(c[0]-npx2//2\*d2,c[0]+npx2//2\*d2,npx2)**

**y = np.linspace(c[1]-npx2//2\*d2,c[1]+npx2//2\*d2,npx2)**

**picture=np.zeros([npx2,npx2])**

**sigma=300**

**nbspot=3**

**tailleimage=12\*int(sigma/d2)**

**picture=np.zeros([npx2,npx2])**

**def gauss2D(x,y,I,sigma):**

**return I\*np.exp(-(x\*\*2+y\*\*2)/(2\*sigma\*\*2))**



```
In = [300,1000,500]
xn = [500,3513,20000]
yn = [10000,4625,15000]
```

```
In = np.asarray(In)
xn = np.asarray(xn)
yn = np.asarray(yn)
```

```
for i in range(npx2):
    for j in range(npx2):
        picture[i,j] = sum(gauss2D(x[j]-xn,y[i]-yn,In,sigma))
```

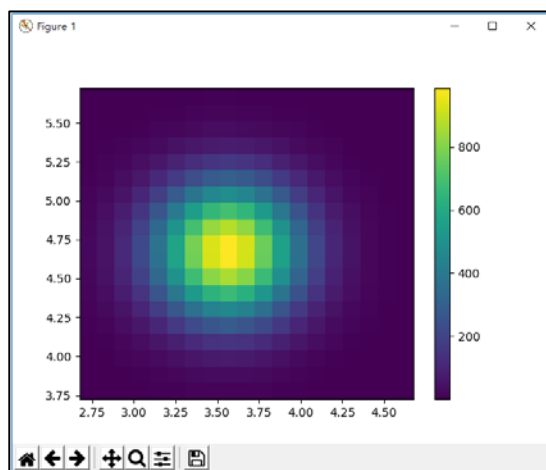
```
N = 100
background = np.ones([npx2,npx2])*N
noisy_background = np.random.normal(background,np.sqrt(background))
noisy_picture = np.random.poisson(picture+noisy_background)
```

```
plt.pcolormesh(x/1000,y/1000,noisy_picture)
##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
plt.colorbar()
```

```
#plt.figure()
#plt.imshow(noisy_picture)
```

```
##plt.title("scan simule")
##plt.xlabel("un pixel=300nm")
##plt.colorbar()
```

```
plt.show()
```



Scan «précis» holo :  
 résolution du scan ( $d_2$ ) = 100nm |  
 nombre de pixel du scan (npx2) = 20

---

/ interactions Python et LabVIEW : *main2.py* /

```
#!/usr/bin/env python
```

```
# coding: utf-8
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
def main2(d2,npx2,X_en_um,Y_en_um):
```

```
    c = [X_en_um,Y_en_um]
```

```
    c = np.asarray(c)*1000
```

```
    x = np.linspace(c[0]-npx2//2*d2,c[0]+npx2//2*d2,npx2)
```

```
    y = np.linspace(c[1]-npx2//2*d2,c[1]+npx2//2*d2,npx2)
```

```
    picture=np.zeros([npx2,npx2])
```

```
    sigma=300
```

```
    nbspot=3
```

```
    tailleimage=12*int(sigma/d2)
```

```
    picture=np.zeros([npx2,npx2])
```

```
def gauss2D(x,y,I,sigma):
```

```
    return I*np.exp(-(x**2+y**2)/(2*sigma**2))
```

```
In = [300,1000,500]
```

```
xn = [500,3513,20000]
```

```
yn = [10000,4625,15000]
```

```
In = np.asarray(In)
```

```
xn = np.asarray(xn)
```

```
yn = np.asarray(yn)
```

```
for i in range(npx2):
```

```
    for j in range(npx2):
```

```
        picture[i,j] = sum(gauss2D(x[j]-xn,y[i]-yn,In,sigma))
```

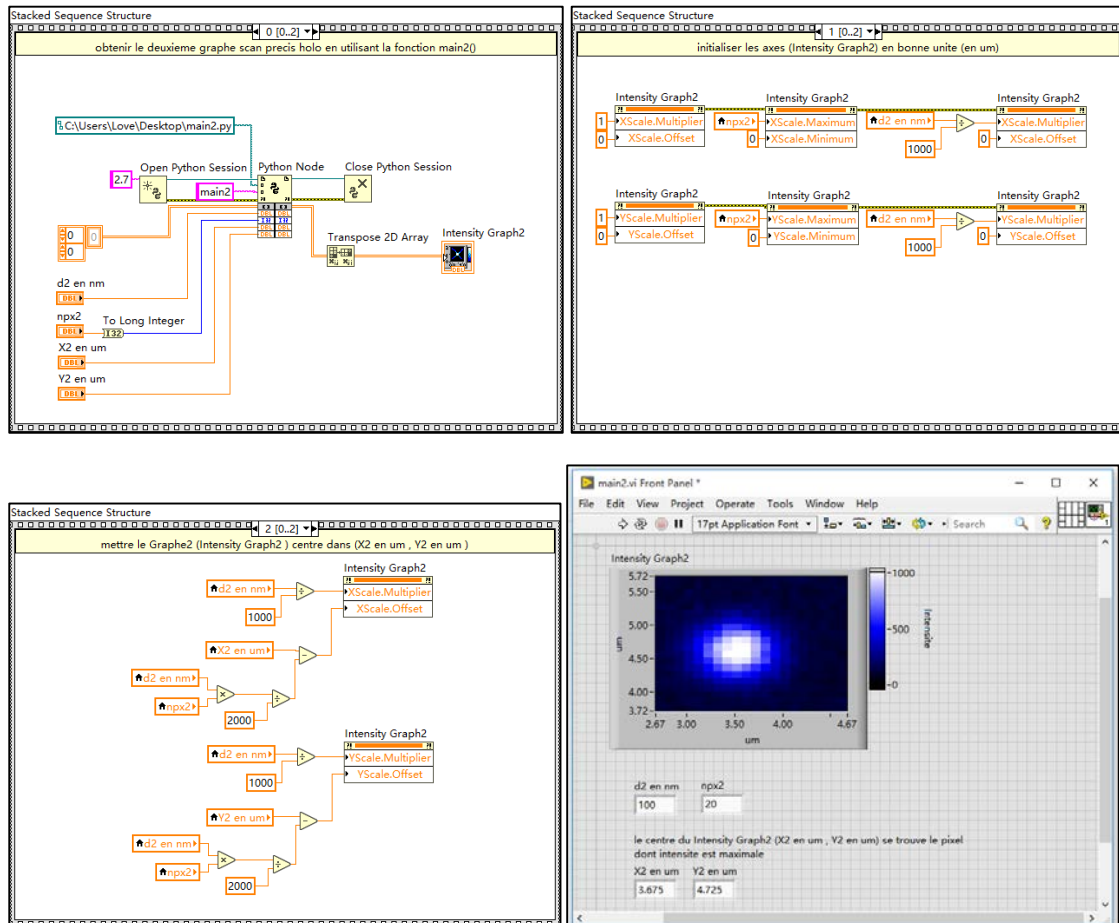
N = 100

background = np.ones([npx2,npx2])\*N

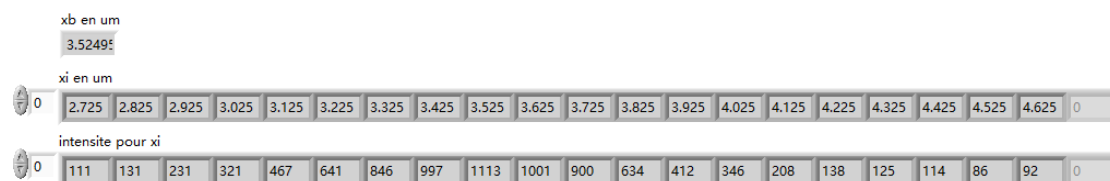
noisy\_background = np.random.normal(background,np.sqrt(background))

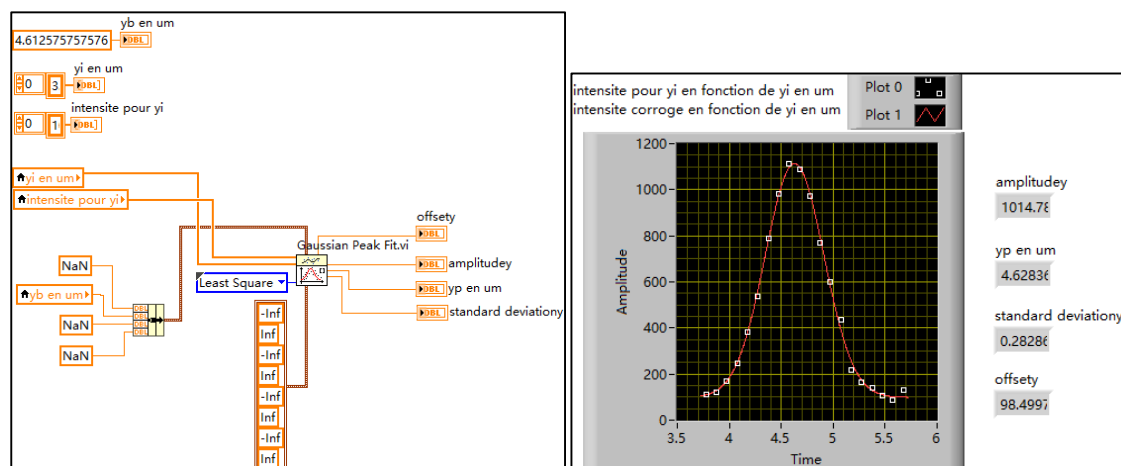
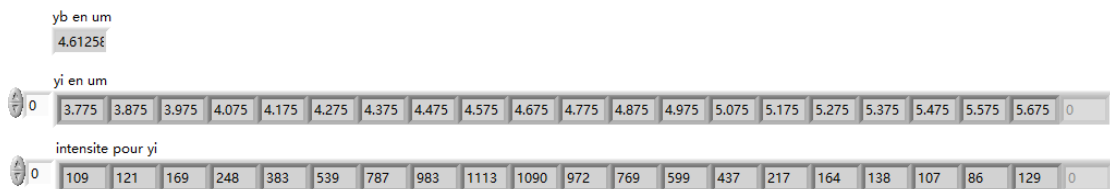
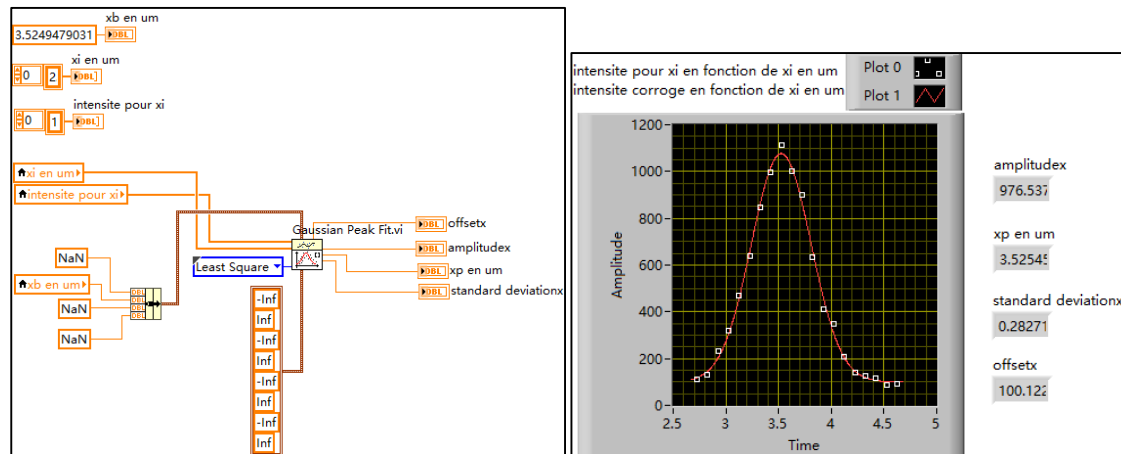
noisy\_picture = np.random.poisson(picture+noisy\_background)

return noisy\_picture.tolist()



## Annexe 5 Ajustement de pic gaussien





## Annexe 6 main.py

```
#!/usr/bin/env python
```

```
# coding: utf-8
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
def main2(d2,npx2,X_en_um,Y_en_um,intensite):
```

```
    c = [X_en_um,Y_en_um]
```

```

c = np.asarray(c)*1000
x = np.linspace(c[0]-npx2//2*d2,c[0]+npx2//2*d2,npx2)
y = np.linspace(c[1]-npx2//2*d2,c[1]+npx2//2*d2,npx2)

picture=np.zeros([npx2,npx2])

sigma=300
nbspot=3

tailleimage=12*int(sigma/d2)
picture=np.zeros([npx2,npx2])

def gauss2D(x,y,I,sigma):
    return I*np.exp(-(x**2+y**2)/(2*sigma**2))

In = [300,intensite,500]
xn = [500,3513,20000]
yn = [10000,4625,15000]

In = np.asarray(In)
xn = np.asarray(xn)
yn = np.asarray(yn)

for i in range(npx2):
    for j in range(npx2):
        picture[i,j] = sum(gauss2D(x[j]-xn,y[i]-yn,In,sigma))

N = 100
background = np.ones([npx2,npx2])*N
noisy_background = np.random.normal(background,np.sqrt(background))
noisy_picture = np.random.poisson(picture+noisy_background)

return noisy_picture.tolist()

```

## List of commands for the USB interface

### *Parameters for the communication port :*

Baud rate : 115200 Bauds      Data bits : 8      Stop bits : 1  
No parity      Flow control : none      Character for transmission end : LF (0xA)

### *List of the commands :*

Important note: the commands have always 5 characters and it is important to respect the syntax of each of those.

#### « \_RAZ\_ » :

This commands sets all the outputs of the DAC to 0V.

#### « INFOS » :

This commands allows to know all the basic informations of the nanopositioning system (name of the stage, number of axis, maximum stroke, etc...)

#### « MOVEX » :

This command allows to move the stage to an exact position (similar function exist for the Y axis and the Z axis : MOVEY and MOVEZ)

As an example, if one wants to move to the 200  $\mu\text{m}$  position (ie 200000 nm), one must write : « MOVEX 200u » or « MOVEX 200000n »