Procédurier

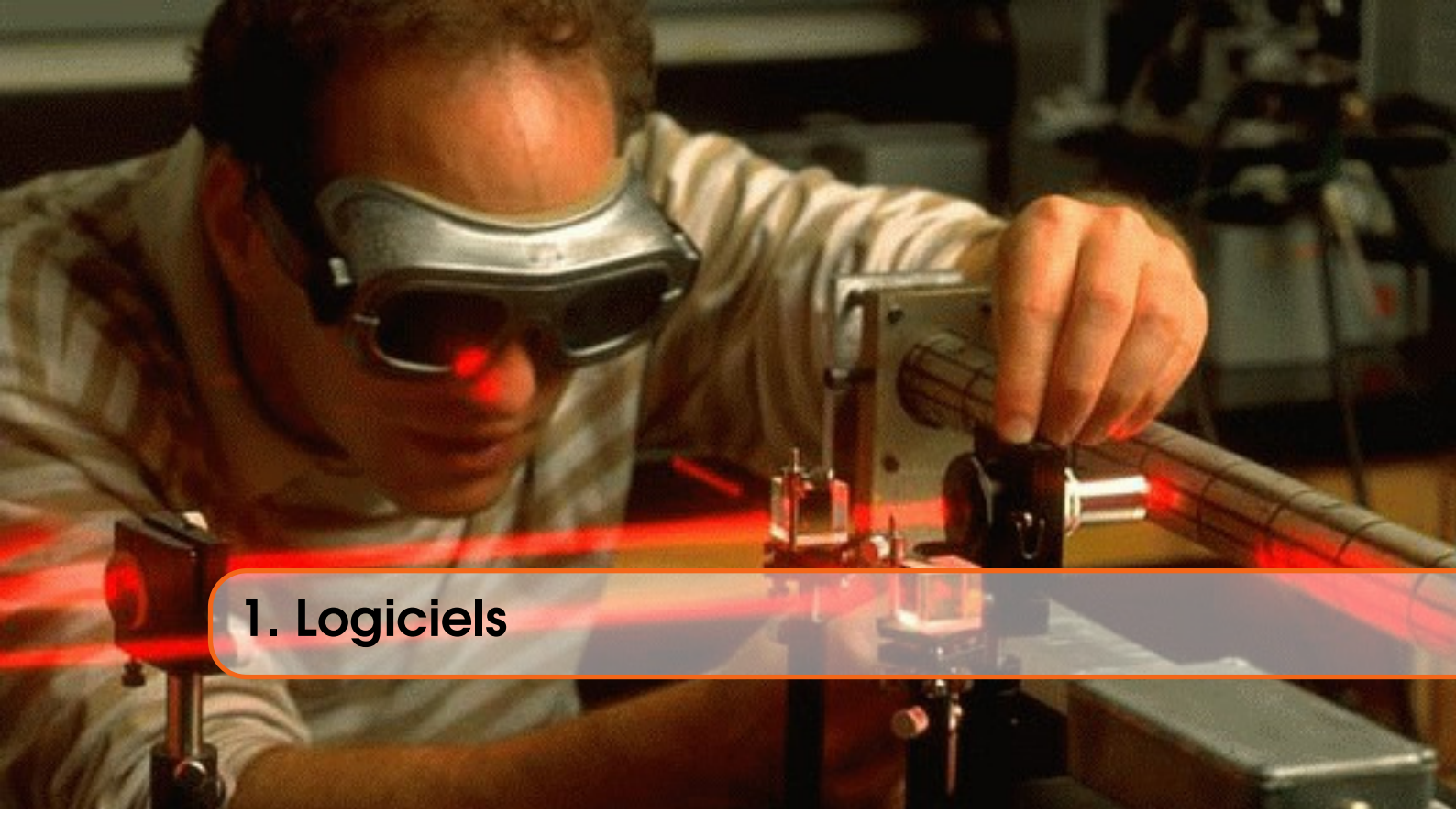
Laboratoire R. Leonelli

Colin-N. Brosseau

https://github.com/ColinBrosseau/LaboratoireLeonelli

Édition 0.2.2

Table des matières



1. Logiciels

De nombreux logiciels ont été mis au point dans le but de contrôler les nombreux appareils et expériences du laboratoire. En général, les expériences et appareils fonctionnant dans le laboratoire *Trivista* utilisent Matlab. De son côté, le laboratoire *U1000* est complètement implémenté avec Python [1].

1.1 Matlab

À moins d'indication contraire, les codes Matlab utilisés au laboratoire ont été testés avec la version 7.1 (R14).

1.1.1 Codes supplémentaires

Plusieurs instruments sont contrôlés par cable GPIB. On utilise deux "modules" associés :

— http ://www.mathworks.com/matlabcentral/fileexchange/216
— http ://www.mathworks.com/matlabcentral/fileexchange/3140

Les deux sont complémentaires. Le premier, de Tom Davis, sert à la communication. Le deuxième, d'Alaa Makdissi, est surtout utilisé pour l'initialisation.

1.2 Python

Plusieurs instruments du laboratoire peuvent être contrôlés avec Python. En général, il s'agit des instruments situés dans le D405 (Labo Raman).

À l'heure actuelle, la manière la plus simple d'installer python sur windows est d'utiliser la **distribution anaconda** https ://www.continuum.io/downloads. Il existe deux branches de Python soient la 2 et la 3. On prend **Python 3**.

1. De nombreux appareils du laboratoire *U1000* ont un interface Matlab, mais aucune mise à jour n'est faite depuis un moment.

1.2.1 Installation sous Windows XP

Pour installer Anaconda, il faut normalement se rendre a l'adresse suivante https ://www.continuum.io/downloads et suivre les instructions.

Cependant, si on installe Anaconda sur Windows XP, il faut utiliser une version plus anciene de Anaconda. Sinon, ca ne fonctionnera pas (les nouvelles versions de Anaconda utilisent Python 3.5 (ou plus) qui ne sont pas compatibles avec Windows XP.) Source : `https://docs.continuum.io/anaconda/#older-versions-of-anaconda`

Sur Windows XP, il faut donc telecharger `https://repo.continuum.io/archive/Anaconda3-2.2.0-Windows-x86.exe`

Installer le programme.

1.2.2 Modules

L'installation de base d'anaconda contient la plupart des modules nécessaires au fonctionnement des scripts utilisés dans le laboratoire. On installe des modules supplémentaires de la façon suivante :

— Démarrer > Anaconda (32-bit) > Anaconda Command Prompt
— Enter la ligne suivante (en adaptant avec le nom du module voulu) dans la console.

Code 1.1 – Installation d'un module Python avec *pip* (dans la console)

```
pip install nom_du_module
```

Les modules nécessaires pour pouvoir utiliser les scripts conçus pour le laboratoire sont énumérés dans la table 1.1. Les modules qui n'ont pas de nom pip sont présents de manière native sous Python.

TABLE 1.1 – Modules nécessaires au fonctionnement des scripts du laboratoire

Module	nom sous pip	Rôle
visa	pyvisa	contrôle des instruments (RS232, GPIB)
struct	-	utilisé par SR830c pour lire le flux binaire
yaml	pyyaml	lire/écrire des fichiers sous format YAML
collections	-	utilisé pour créer des dictionnaires ordonnés
pandas	pandas	manipulation simplifiée des structures des données scientifiques
matplotlib	matplotlib	tracer des graphiques
lmfit	lmfit	ajustement non linéaire (moindre carrées)

Code 1.2 – Installation des packets supplémentaires (console)

```
conda install pyserial
conda install comtypes
pip install pyvisa
pip install lmfit
pip install pyyaml
pip install pandas
pip install matplotlib
# collections ???
# struct ???
```

visa

Le module *visa*[2] permet d'interagir facilement avec les instruments du laboratoire. Il nécessite que *LabView* soit installé sur l'ordinateur (??? à confirmer). Le Code 1.3 montre un exemple minimal de son utilisation.

Code 1.3 – Exemple d'utilisation de *visa*

```
import visa  # importe le module
rm = visa.ResourceManager()

# affiche la liste des appareils connectes
rm.list_resources()
# ex. de sortie:
#     ('ASRL1::INSTR', 'ASRL2::INSTR', 'GPIB0::12::INSTR')

# initialise la communication
inst = rm.open_resource('GPIB0::12::INSTR')  # appareil connecte
    sur le permier interface GPIB a l'adresse 12

# questionne l'appareil
print(inst.query("*IDN?"))  # indenfication de l'appareil (
    commande typique)

# Ce qui suit est equivalent a query, mais en deux etapes

# ecrit dans l'appareil
inst.write("*IDN?")

# lit la reponse de l'appareil
print(inst.read("*IDN?"))
```

Matplotlib

??? faire un mini crash course ici

Pour un tutoriel sur l'utilisation de Matplotlib, se référer à l'adresse `http://matplotlib.org/users/pyplot_tutorial.html`.

Pandas

??? faire un mini crash course ici

`http://pandas.pydata.org/pandas-docs/stable/10min.html#min`

`http://nbviewer.jupyter.org/github/jvns/pandas-cookbook/blob/v0.1/cookbook/Chapter%201%20-%20Reading%20from%20a%20CSV.ipynb`

Interface pour Winspec

L'interface Python pour Winspec doit être construite pour pouvoir accéder à Winspec par Python. Cette précédure est à faire une seule fois et consiste en deux étapes successives

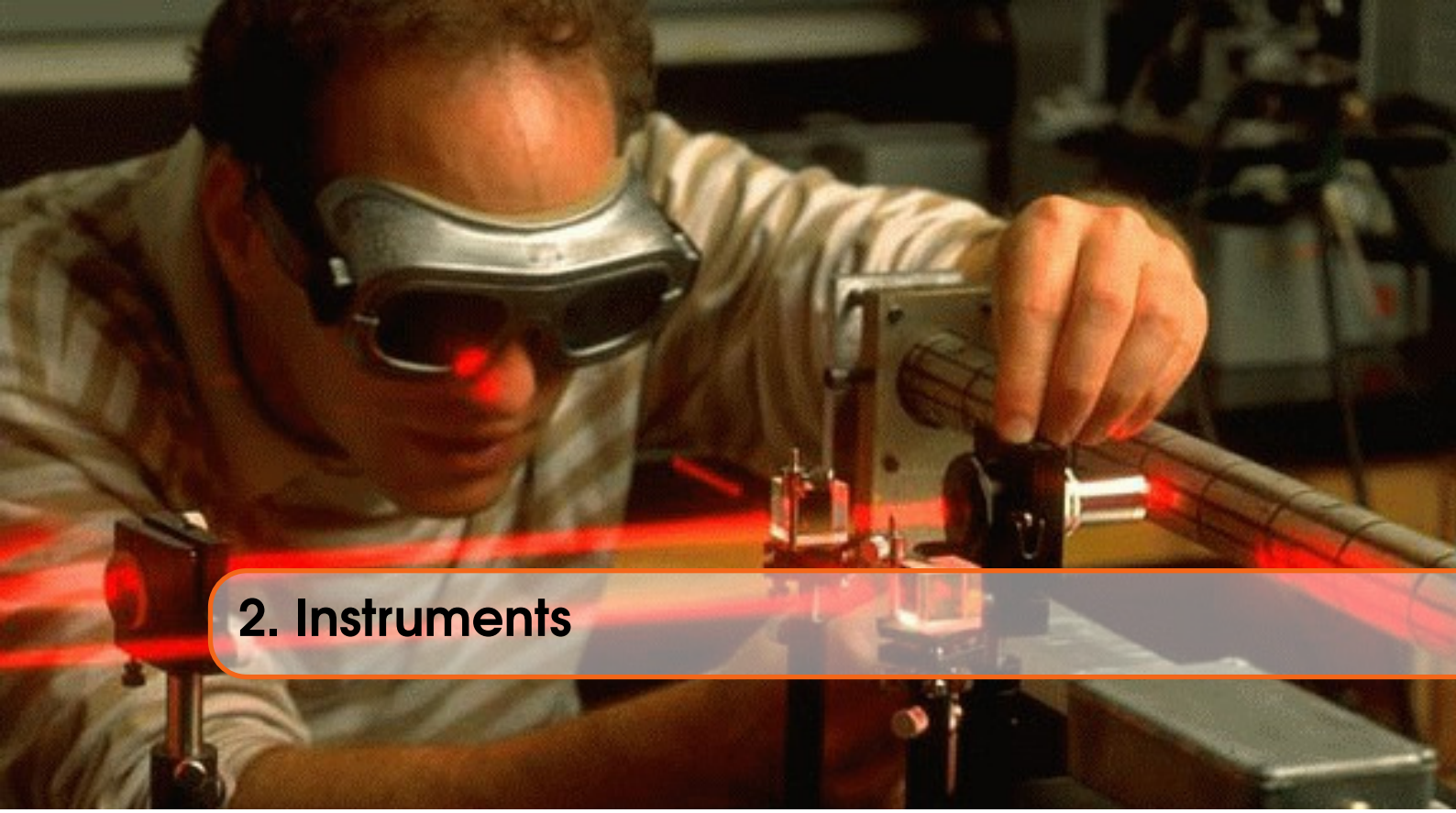
Code 1.4 – Creer l'interface pour Winspec (console)

```
ipython makepy.py
```

2. `https://pyvisa.readthedocs.io/en/stable/`

Code 1.5 – Création de l'interface Python pour Winspec

```
import comtypes.client as cc
# This line has to be before import comtypes.gen.WINX32Lib as
    WinSpecLib (at least on the first run.)
# It creates the file C:\Program Files\Anaconda3\lib\site-
    packages\comtypes\gen\
    _1A762221_D8BA_11CF_AFC2_508201C10000_0_3_11.py
# It allows to import comtypes.gen.WINX32Lib
cc.GetModule( ('{1A762221-D8BA-11CF-AFC2-508201C10000}', 3, 11))
     # comes from regedit (look for IWinx32App2), major version,
    minor version
```



2. Instruments

2.1 Bomem

2.1.1 Démarrage

- Allumer le Bomem (interrupteur en bas en arrière).
- Allumer le processeur vectoriel (DA-PC).
- Démarrer le logiciel BOMEM PCDA.
- Lorsque l'affichage du processeur vectoriel passe à 6, cliquer sur OK dans la fenètre du logiciel.
- Charger une configuration appropriée pour le détecteur/lame séparatrice (Experiment>Open...).
- Dans la configuration, spécifier
  - de quel côté se trouve le détecteur (gauche ou droite)
  - quelle est l'entrée utilisée par le détecteur 1,2,3, etc.
- Connecteur la sortie de l'amplificateur du détecteur dans l'entrée spécifiée.

Configuration relative aux détecteur

La nature du détecteur influence les paramètres de la mesure. La table 2.1 indique les paramètres recommandés en fonction du détecteur utilisé.

TABLE 2.1 – Configuration du Bomem relative au détecteur

	Si	MCT	InSb	InGaAs	Ge (vis)
Source	2000 - 25000	2000 - 25000	2000 - 25000	2000 - 25000	2000 - 25000
Beamsplitter	4000 - 25000	1200 - 10000	2000-25000	4000 - 25000	4000 - 25000
Filter	0 - 63192	0 - 63192	0 - 63192	0 - 63192	0 - 63192
Detector	9999 - 25000	2000 - 8000	3000 - 14000	5000 - 12000	5000 - 13000
Time Delay (us)	1.6	4	4	0.1	160
Speed (cm/s)	0.5	0.15	0.05	0.1	0.5

Gain du détecteur

Il est possible d'appliquer deux gains sur le signal du détecteur. Ces paremètres sont ajustés juste avant la mesure, lorsqu'on clique sur *Raw Spectrum*. Entre la frange 1 et *Fringe #* le signal du détecteur est amplifié d'un facteur *Gain Base*. Le signal suivant les franges *Fringe #* sont amplifiées d'un facteur *Gain Pos*.

(???à confirmer auprès de Richard) Seul les détecteurs utilisant les amplificateurs Bomem (dans notre cas le détecteur silicium) prennent en compte le *Gain Pos*.

Si de fortes oscillations se produisent dans l'interférogramme et/ou le spectre il faut mettre *Gain Pos* à *None*. [1]

2.1.2 Logiciel du Bomem

Dans le logiciel du Bomem, le bouton %ADC permet d'avoir une idée de l'intensité du signal. Le détecteur sature lorsqu'il indique **>100**. Lorsque le détecteur est complètement saturé, il indique **0**.

2.1.3 Transfert des données de l'ordinateur du Bomem vers l'extérieur

Pour transférer des données à l'extérieur de l'ordinateur de contrôle du Bomem, il y a deux possibilités [2] :

- disquettes
- connection réseau [3]
  - l'ordinateur du Bomem apparait sur le réseau PMC sous `\\Bomem333`. Les données se trouvent dans le répertoire *DONNEES*.

2.1.4 Transfert des mesures vers Matlab

On utilise la fonction `bomem.m` pour convertir les données binaires prises par le logiciel en variables Matlab. Code 2.1 montre comment utiliser cette fonction.

Code 2.1 – Lecture des données brutes du Bomem avec Matlab

```
[x,y] = bomem(fichier_sans_extension_spc, unite, calibration)

% fichier_sans_extension_spc
%     nom du fichier.spc (on ne doit pas mettre l'extension .
   spc)

% unite:
%     0 cm-1
%     1 eV
%     2 A

% calibration
%     0 pas de calibration
%     1 Si - lame quartz
%     2 InGaAs - lame quartz
%     3 Ge - lame CaF2
```

1. Il est aussi possible que les oscillations dans l'interférogramme soient causées par l'excitation laser. Dans ce cas, il faut la diminuer, en général à l'aide d'un filtre coloré.
2. Les clés USB ne fonctionnent pas sur cet ordinateur.
3. L'ordinateur de contrôle du Bomem est très vieux et ne doit jamais être laissé connecté au réseau en permanence.

```
%      4 InSb - lame quartz
%      5 InSb - lame CaF2

% x
%      position spectrale (depend de unite)
% y
%      intensite mesuree
```

2.1.5 Détecteurs

Le Bomem possède un filtre DC en entrée. On peut donc utiliser soit un signal DC soit un signal AC.

Détecteur Silicium

Le détecteur Silicium est utilisé pour les mesures faites dans le visible. Il utilise l'amplificateur installé sur le côté de son support.

Détecteur InGaAs

Le détecteur InGaAs [4] est sensible dans la gamme 1000 - 1900 nm [5].

Ce détecteur a besoin d'azote liquide pour fonctionner. Il est recommandé de pomper le vide d'isolation tous les 6 mois.

Le signal de ce détecteur est amplifié [6]. Pour l'utilisation avec le Bomem, on utilise la sortie DC (premier étage).

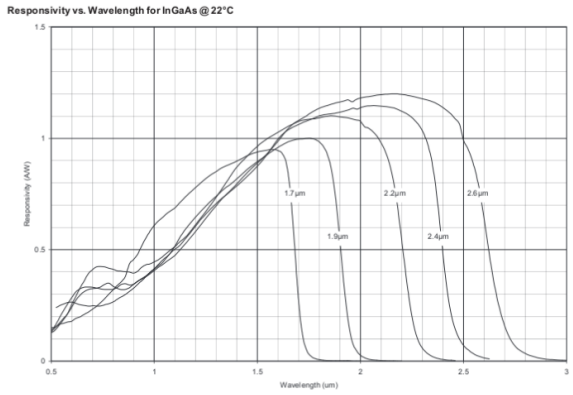


FIGURE 2.1 – Sensibilité détecteur Bomem InGaAs (à 20 °C)

Détecteur InSb

Le détecteur InSb [7] est sensible dans la gamme 1000-2800 nm.

Ce détecteur a besoin d'azote liquide pour fonctionner. Il est recommandé de pomper le vide d'isolation tous les 6 mois.

Le signal de ce détecteur est amplifié [8]. Pour l'utilisation avec le Bomem, on utilise la sortie DC (premier étage).

4. Numéro de modèle : J23D-M205-R01M-60-1.9, http ://www.teledynejudson.com/files/pdf/ingaas_PB4206.pdf
5. La longueur d'onde maximale détectée dépend de la température. Elle est de 1900 nm à 20 °C) mais passe à 1810 nm à -85 °C. Une estimation rapide porte la longueur d'onde maximale à ≈ 1750 nm à température de l'azote liquide.
6. Amplificateur PA-7-70, RD=70 GΩ, CD=870 pF, RS=10.36 Ω
7. Numéro de modèle : J10D-M204-R01M-60-SP28, http ://www.judsontechnologies.com/files/pdf/InSb_shortform_Mar2003.pdf
8. Amplificateur PA-9, RD=150 MΩ, CD=355 pF

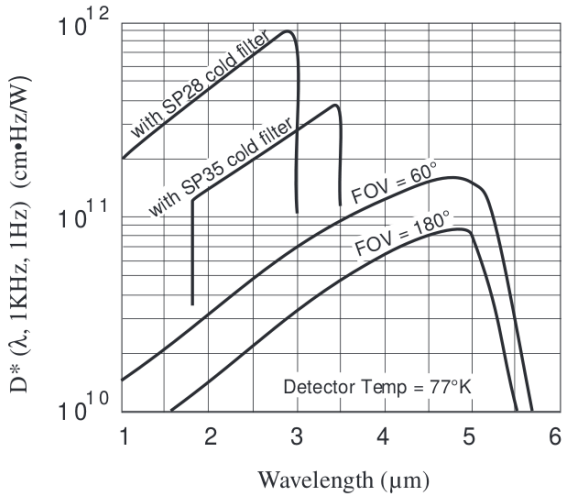


FIGURE 2.2 – Sensibilité détecteur Bomem InSb (à 77 K)

2.2 Caméras Princeton Instruments

Les caméras Princeton Instruments (la PDA InGaAs et la Pixis) sont contrôlées soit par le logiciel Winspec ou par Matlab. Les deux sont mutuellement exclusifs à un instant donné.

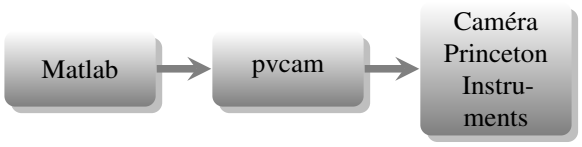
Lorsqu'on change de détecteur (passant de la PDA à la Pixis ou inversement), il **faut** redémarrer Winspec et faire la procédure de détection. Parfois ca ne fonctionne pas : il faut débrancher le cable USB du Trivista et recommencer. Il arrive également qu'on doive redémarrer complètement l'ordinateur

2.2.1 Winspec

Winspec permet de vérifier que la caméra fonctionne et tester rapidement le signal. Il n'y a pas de calibration Pixel → Longueur d'onde.

2.2.2 Matlab

Matlab peut contrôler les caméras Princeton Instruments via la librairie pvcam [9] :



Code 2.2 – Exemple d'utilisation des caméras Princeton Instruments avec Matlab

```
% initialise la communication
camera_PrincetonInstruments('initialise')

% temperature cible du detecteur (C)
Camera_PrincetonInstruments('TEMP_SETPOINT')  % lit
Camera_PrincetonInstruments('TEMP_SETPOINT', -70)  % ecrit
```

9. Le logiciel PVCAM est disponible à l'adresse `http://ftp.piacton.com/Public/Software/Official/PVCAM/pvcam32.exe`. La documentation relative à PVCAM se trouve à `http://ftp.piacton.com/Public/Manuals/Princeton%20Instruments/PVCAM%202.7%20Software%20User%20Manual.pdf`

```
% temperature reele du detecteur (C)
Camera_PrincetonInstruments('TEMP')  % lit

% nom de la camera (tel que vu par Winspec)
Camera_PrincetonInstruments('nom')
% nom du detecteur CCD
Camera_PrincetonInstruments('CHIP_NAME')

% lit/ecrit le gain
Camera_PrincetonInstruments('GAIN')
Camera_PrincetonInstruments('GAIN', 2)
% lit/ecrir la vitesse de l'ADC
Camera_PrincetonInstruments('SPEED')
Camera_PrincetonInstruments('SPEED','100 kHz')

% taille (en pixels) du detecteur
[xsize, ysize] = Camera_PrincetonInstruments('
   DIMENSION_DETECTEUR')

% obturateur mecanique
Camera_PrincetonInstruments('SHUTTER','ferme')  % ferme
Camera_PrincetonInstruments('SHUTTER','ouvert')  % ouvre

% mesure un spectre (somme ``verticalement'') sur toute la
   plage disponible
PARAM.window = struct
PARAM.window.s1 = 0;  % premier pixel ``horizontal'' (0-based)
PARAM.window.s2 = xsize - 1;  % dernier pixel ``horizontal''
   (0-based)
PARAM.window.sbin = 1;  % somme (bin) ``horizontal''
PARAM.window.p1 = 10;  % premier pixel ``vertical'' (0-based)
PARAM.window.p2 = ysize - 5;  % dernier pixel ``vertical'' (0-
   based)
% Le parametre PARAM.window.sbin est calcule automatiquement
   pour la fonction 'spectre'
PARAM.t_acc = 5; % 5 secondes d'acquisition
PARAM.n_acc = 2; % 2 acquisitions
data = camera_PrincetonInstruments('spectre', PARAM) % prend
   la mesure
plot(data)

% mesure une image simple (tout le detecteur, 1 seconde d'
   acquisition)
data = camera_PrincetonInstruments('image');

% mesure une image, somme ``verticalement'', acquisition de 2
   secondes
PARAM.window  % conserve les parametre precedents
PARAM.window.pbin = 5  % somme ``verticalement'' 5 pixels
```

```
PARAM.t_acc=2  % acquisition de 2 secondes
data = camera_PrincetonInstruments('image',PARAM);

% Lit des parametres arbitraires (pas definis dans la fonction
    Camera_PrincetonInstruments.m)
% Pour la liste et la signification des parametres de PVCAM,
   se referer a
%  http://ftp.piacton.com/Public/Manuals/Princeton\%20
   Instruments/PVCAM\%202.7\%20Software\%20User\%20Manual.pdf
% distance 'horizontale' entre les pixels
Camera_PrincetonInstruments('GET_PARAMETER', '
   PARAM_PIX_SER_DIST')
% nombre de bits valides dans les donnees retournees
Camera_PrincetonInstruments('GET_PARAMETER', 'PARAM_BIT_DEPTH'
   )

% De maniere similaire que 'GET_PARAMETER', on peut ecrire un
   parametre avec 'SET_PARAMETER'.

% distance 'horizontale' et 'verticale' entre les pixels (
   utile pour la calibration pixel -> longueur d'onde) (centre
    a centre, nm)
[x,y] = Camera_PrincetonInstruments('PIXEL_DIMENSION')

% ferme la communication
Camera_PrincetonInstruments('fermer')
```

2.2.3 PDA InGaAs

Ce détecteur[10] (OMA V :1024-2.2 LN??? à vérifier???) est une rangée de 1024 pixels sensibles jusqu'à 2.2 μm (à température pièce, la limite à basse température serait d'environ 2.1 μm).

Ce détecteur a besoin d'azote liquide pour fonctionner et on stabilise la température à -100 °C. Il **doit** être connecté à son contrôleur (allumé) lorsqu'il y a de l'azote dans le réservoir.

Il est recommandé de pomper le vide d'isolation tous les 6 mois.

Étant donné qu'il est sensible à l'infrarouge, le bruit thermique est particulièrement important sur ce détecteur. En général, on mesure 30 000 comptes *noirs* par seconde. Étant donné que ce détecteur sature à 64 000 comptes, on limite l'accumulation à une seconde.

On utilise la sortie $\overline{SCAN}$ du contrôleur pour l'obturateur mécanique.

On peut contrôler ce détecteur avec la fonction Matlab générique `Camera_PrincetonInstruments.m`.

2.2.4 Pixis

La caméra Pixis (Pixis 256[11]) est composé de 1024x256 pixels sensibles jusqu'à ≈ 1 μm. Elle est refroidie par effet Pelletier à -70 °C.

Cette caméra a un bruit de lecture d'environ 600 comptes et un bruit noir suffisamment bas pour être négligeable dans la plupart des cas.

10. ftp://ftp.princetoninstruments.com/public/Manuals/Princeton%20Instruments/OMAV%20InGaAs%20System%20Manual.pdf
11. ftp://ftp.princetoninstruments.com/public/Manuals/Princeton%20Instruments/PIXIS%20System%20Manual.pdf

FIGURE 2.3 – PDA InGaAs

On utilise la sortie LOGIC OUT du détecteur pour l'obturateur mécanique.

Tout comme la PDA InGaAS, on peut contrôler ce détecteur avec la fonction Matlab générique `Camera_PrincetonInstruments.m`.

FIGURE 2.4 – Pixis

2.3 CCD Raman

La caméra CCD fixée sur le U1000 est composé de 1340x400 pixels sensibles jusqu'à $\approx 1\ \mu$m.

Cette caméra est refroidie par effet Pelletier à -100 °C. Elle **doit** être connectée à son contrôleur (allumé) lorsqu'il y a de l'azote dans le réservoir.

Il est recommandé de pomper le vide d'isolation tous les 6 mois.

Ce détecteur a besoin de Winspec32 pour pouvoir être contrôlée. En plus de l'acquisition manuelle par Winspec, il est possible de le faire avec Python. Cette dernière option permet d'automatiser l'acquisition.

2.3.1 Python

Le contrôle de la caméra par Python fonctionne selon le schéma suivant :

Il est donc nécessaire que Winspec soit en fonction lors de cette opération. Un exemple minimal est donné dans Code 2.3.

Code 2.3 – Exemple d'utilisation de Winspec avec Python

```
import WinspecCOM
camera = WinspecCOM.winspec()

camera.initialConfiguration()  # set most common values in cosmic
    ray and temperature

camera.setTemperature(-100)  # set the target temperature to -100
    C
camera.setTemperature()[0]  # read the target temperature
camera.actualTemperature()[0]  # read the actual temperature

# ROI (xmin, xmax, xgroup, ymin, ymax, ygroup)
camera.ROI(1, [1, 1340, 1, 1, 400, 400])  # set (for come reason,
    one have to manually confirm that in Winspec)
camera.ROI()  # read

# exposure unit 'ms', 's', 'min', 'h'
camera.exposureUnit('s')  # set
camera.exposureUnit()[0]  # read

# exposure time
camera.exposure(1.5)  # set
camera.exposure()[0]  # read

# number of accumulations
camera.nAccumulations(2)  # set
camera.nAccumulations()[0]  # read

# ADC speed '100 kHz', '1 MHz'
camera.adcSpeed('100 kHz')
camera.adcSpeed()[0]

# background removal True, False
camera.removeBackground(True)  # set
camera.removeBackground()[0]  # read

# cosmic ray mode 'off', 'temporal', 'spatial'
camera.cosmicMode('spatial')  # set
camera.cosmicMode()[0]  # read
```

```
# cosmic ray sensitivity
camera.cosmicSensitivity(50)  # set
camera.cosmicSensitivity()[0]  # read

camera.acquireBackground()  # measure the background

# measurements
x, y, filename = camera.measureSimple()  # default values:
    accumulation for 1 second, one image and one accumulation
x, y, filename = camera.measureSimple(exposureTime=2)  #
    accumulation for 2 second2, default values: one image and one
    accumulation
# x and y contains position/intensity values
# filename is name of the saved file (.SPE format)

camera.getParamAll('test.txt')  # save current values in a file (
    nice for development)

camera.stop()  # stop current measurement

camera.isRunning()[0]  # check if camera is currently measuring
```

2.4 HydraHarp

Le Hydra Harp (figure 2.5) est utilisé pour les mesures de comptage de photons corrélés.

Notre module comporte deux canaux d'entré en plus de la synchronisation. Il est important de ne pas dépasser 1.5 Volt en entrée sur les canaux.

Les spécification et le logiciel de contrôle sont disponibles à l'adresse suivante : `http://www.picoquant.com/products/category/tcspc-and-time-tagging-modules/hydraharp-400-multichannel-`

La boite de contrôle a une résolution temporelle de 1 ps. Cependant, la réponse instrumentale des détecteurs (SPAD) limite la résolution effective à environ 200 ps.

2.4.1 Détecteur

Les détecteurs [12] (2) que nous possédons, sont des SPAD de marque MPD (micro photon devices) dans la série PDM. Ils ont les caractéristiques suivantes : 50 μm, cooled, timing, Grade C. Ils doivent être alimentés pour fonctionner.

Il y a trois sorties sur les détecteurs : *TTL Out, Timing Out, Gate In*. Pour les mesures en comptage de photon, nous utilisons la sortie *Timing Out*

Leur sensibilité en fonction de la longueur d'onde est illustrée à la figure 2.6.

After-pulses

Ces détecteurs ont environ 3% d'after-pulses (Pascal Grégoire a mesuré 2.5%). Il s'agit de pulses présents environ 100 ns après le maximum.

Il est possible [13] de réduire considérablement la présence d'after-pulses en utilisant l'entrée *gate in* du SPAD.

12. Fiche technique `https://www.picoquant.com/images/uploads/downloads/pdm_series.pdf`
13. `http://www.tandfonline.com/doi/abs/10.1080/09500340.2012.698659`

FIGURE 2.5 – HydraHarp et détecteur SPAD

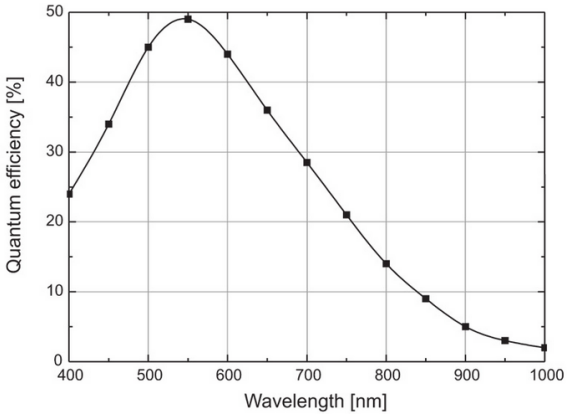


FIGURE 2.6 – Sensibilité spectrale des détecteurs SPAD

2.4.2 Logiciel du constructeur

???à compléter

FluoFit

2.4.3 Matlab

Le code Matlab présenté ici permet de mesurer en mode comptage de photon corrélé. Dans l'état actuel, il ne permet pas d'accéder aux modes avancés (TTTR (enregistrement individuel temporel de tous les événements sur tous les canaux : T2 et T3).

???

Modifier le code ci-bas pour réfléter la nouvelle version. 'so' et 'sl' sont maintenant joints dans 'slevels' ???

Code 2.4 – Exemple d'utilisation de HydraHarp avec Matlab

```
% Initialise la communication et met le detecteur dans un
   mode predefini
hydra('initialise')
% Mesure avec les parametres suivants:
%   Cree un fichier .dat (ascii) en plus du fichier .mat
%   sync offset = 3 ps
%   sync zero = 9 mV
%   resolution = 8 ps
%   input channel 1 trig level = 599 mV
%   temps acquisition = 1 seconde (defaut)
%   fichier de sortie = YYYYMMDDThhmmss.mat suivant la date
   et l'heure du debut de la mesure (defaut)
%           par exemple 20160824T154223.mat pour une mesure
    faite le 24 aout 2016 a 15h42 et 23 secondes.
[x,y] = hydra('measure', 'DAT', 'so', 3, 'sz',9, 'r',8 ,'il'
   ,1,599);
% ferme la communication
hydra('stop' )
% taux de comptage des entrees
[countRateSync, countRateInputs] = hydra('TAUX' )
% place le detecteur dans un mode prefefini
hydra('BASECONFIG')
```

2.5 Lockin Zurich Instruments

2.5.1 Matlab

Code 2.5 – Exemple d'utilisation du lockin Surich Instruments avec Matlab

```
???
```

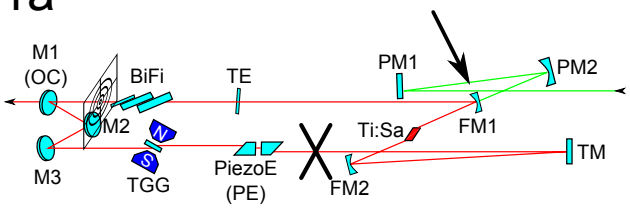
2.6 Matisse

Le Matisse (figure 2.7) est un laser de longueur d'onde ajustable dans une plage d'environ 300 nm (Ti :Sa) et 100 nm (Colorant). La puissance de sortie est de l'ordre de 2 watts.

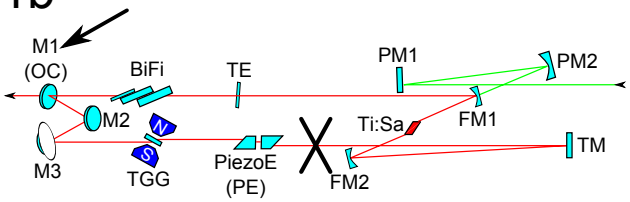
Chaque ensemble de mirroir permet de couvrir une plage d'environ 100 nm (voir tableau 2.6).

Ensemble	Plage utile (nm)	Milieur de gain
MOS-1	700 - 780	Ti :Sa
MOS-2	750 - 870	Ti :Sa
MOS-3	860 - 1015	Ti :Sa
MOS-4	550 - 660	Colorant

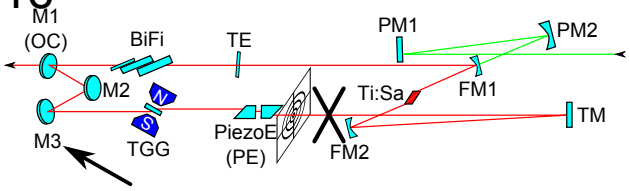
1a



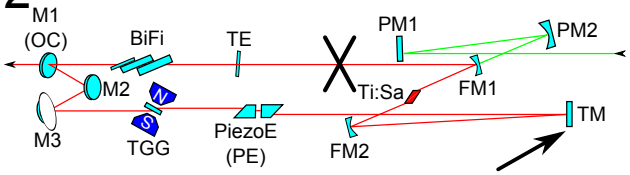
1b



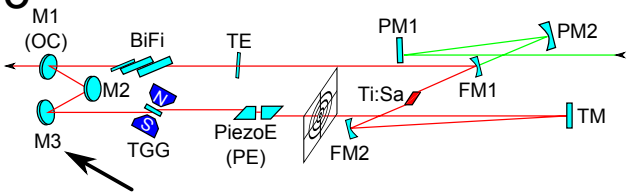
1c



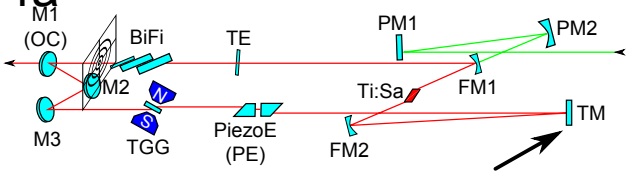
2



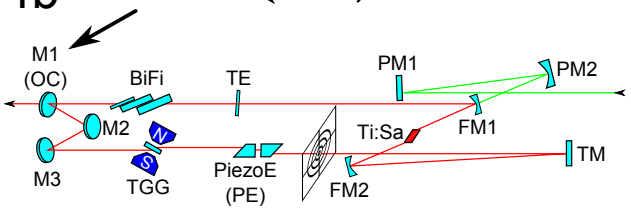
3



4a



4b



	Ti :Sa	Colorant
Spatial Mode	TEM_{00}	
Beam Radius	0.7 mm (typical)	
Beam Divergence	< 1 mrad (half angle)	
Linewidth	< 10 MHz rms (without stabilisation)	< 20 MHz rms (without stabilisation)
	≈ 30 kHz rms (with TS stabilisation)	< 50 kHz rms (with TS stabilisation)
Amplitude Noise	< 0.25% rms (TS)	
Beam Polarization	horizontal	

Plusieurs documents d'entretien du Matisse non disponibles sur l'Internet sont regroupés dans le chapitre 4.

On peut contrôler le Matisse avec le logiciel du constructeur où avec Python. Les deux manières étant mutuellement exclusives [14].

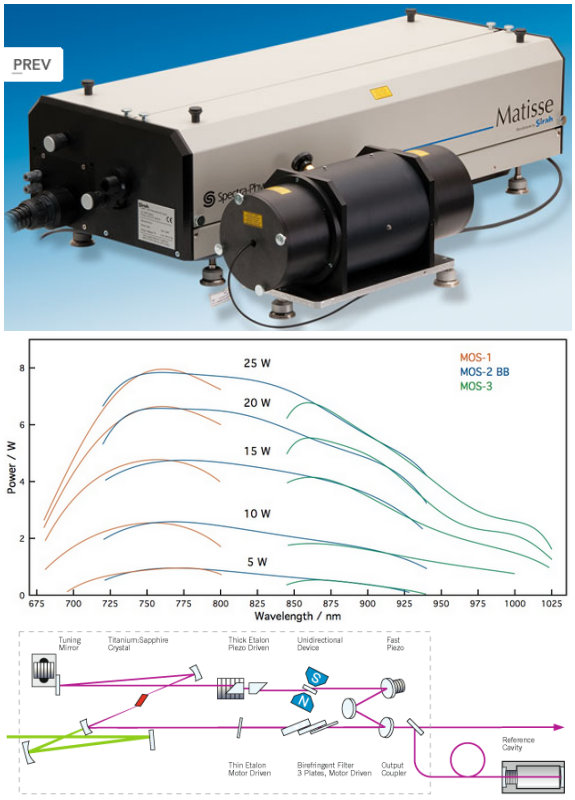


FIGURE 2.7 – Matisse et sa cellule de référence, Plage d'utilisation des ensembles de mirroirs (Ti :Sa), Schémas optique de la cavité

2.6.1 Logiciel du contructeur

Étant donné des choix de conception de Sirah, chaque Matisse doit être contrôlé avec son propre logiciel [15]. On ne peut pas interchanger les deux lasers parce que le logiciel stoque la configuration de son laser attitré. Pour une raison apparentée, on ne peut pas contrôler les deux lasers avec le

14. Les deux peuvent sembler fonctionner ensemble pendant un long moment, mais on aura d'éventuelles erreurs dans le code Python.

15. `Téléchargement:http://www.sirah.com/laser/cw-ring-lasers/matisse-commander-software`

même ordinateur [16].

La version du logiciel utilisé doit correspondre à celle du microgiciel dans le contrôleur du laser.

2.6.2 Python

La classe Matisse sert à interagir avec le laser Matisse. On peut contrôler les deux Matisses (éventuellement en même temps) avec Python. Voir Code 2.6 pour un exemple d'utilisation.

Code 2.6 – Exemple d'utilisation du Matisse avec Python

```
# importe le module
import Matisse

??? tests a ajuster !!! ???
class
xrange
for a = 't':

if

% toto
```

2.6.3 Wavemeter

Les Matisse sont équipés de *wavemeter* permettant de lire la longueur d'onde de sortie. La section 2.16 explique comment lire la longueur d'onde à partir de Python.

2.7 Mira

Le Mira est pompé par le Verdi. Ce laser Ti :Saphire est utilisable en mode continu et pulsé.

Démarrage

1. Positionner en mode *CW* (contrôleur du Mira)
2. Pomper avec le Verdi à pleine puissance (10 W)
3. Attentre au moins 30 minutes
4. Touner le *Biréfringent* pour sélectionner la longueur d'onde désirée
5. Ouvrir complètement la fente de sortie
6. Optimiser la puissance

Mode pulsé

1. Procéder au démarrage
2. Fermer la fente de sortie de manière à avoir la moitié de la puissance maximale
3. Optimiser la puissance avec la position latérale de la fente de sortie
4. Répéter 2 et 3 jusqu'à convergence
5. Positionner en mode *beta lock*
6. Surveiller la trace sur l'oscilloscope
7. Ouvrir lentement la fente de sortie jusqu'à obtenir le mode pulsé (visible sur l'oscilloscope

16. Le code Python permet de contrôler les deux lasers en même temps.

Ajustement de la longueur d'onde

La longueur d'onde du Mira est déterminée par le *Biréfringent*. Le tableau 2.2 montre la correspondance mesurée en 2011.

TABLE 2.2 – Correspondance entre la position de la vis et la longueur d'onde de sortie pour le laser Mira. (Valide en 2011-2017)

Vis	Longueur d'onde (nm)
5.90	710
5.75	723
5.50	744
5.25	766
5.00	790
4.75	814
4.50	839
4.25	866
4.00	892
3.75	922

2.8 Millenia

Le Millenia (figure 2.8) sert à pomper le Matisse.

Contrôle de la puissance

La façon la plus simple d'utiliser ce laser est en mode manuel où les parametres sont ajustés au moyen de la boite de contrôle. On peut également contrôler

FIGURE 2.8 – Millenia

2.9 Modulateurs Acousto-Optique

Les figures 2.9-2.13 montrent les spécifications des modulateurs.

ACOUSTO-OPTIC MODULATOR

MODEL NUMBER: **FQM-80-20-.400**
SERIAL NUMBER: **1310-AO-13006**

	SPECIFICATIONS	TEST DATA
Substrate	UV Grade Fused Silica, Brewster cut	UV Grade Fused Silica, Brewster cut
Maximum Optical power	100 W/mm^2	100 W/mm^2
Wavelength	400 nm	488 nm
Center Frequency	80 MHz	80 MHz
Active Aperture	1 mm	1 mm
Beam Diameter Inside Crystal	0.2 mm	0.2 mm
Rise Time	30 nsec	30 nsec
Digital Modulation Bandwidth	20 MHz	20 MHz
Optical Transmission	> 98%	> 98%
Diffraction Efficiency	~ 60 %	40%. @ 488 nm meas. ~ 60 @ 400 nm calc.
Bragg Angle	3 mrad	3 mrad
Separation Angle	5 mrad	5 mrad
Wave Front Distortion	λ/10	λ/10
Extinction ratio	> 1000:1	> 1000:1
Acoustic Velocity	5.96+3 m/sec	5.96+3 m/sec
RF Power #	~ 3 Watts	~ 3 Watts #
Input Impedance	50 Ohms	50 Ohms
V.S.W.R.	2.:1	2.:1
Case Type	Air Cooled	Air Cooled
Optical Polarization	Linear	Vertical
RF Connector	SMA	SMA

Device is subject to RF saturation; that is, increasing RF may lead to worse performance. Shorter wavelengths tend to saturate at lower RF powers.

Device must be rotated about an axis parallel to SMA to meet Brewster angle, in addition to rotating about an axis perpendicular to axis of SMA for Bragg angle.

FIGURE 2.9 – Modulateurs Acousto-Optique

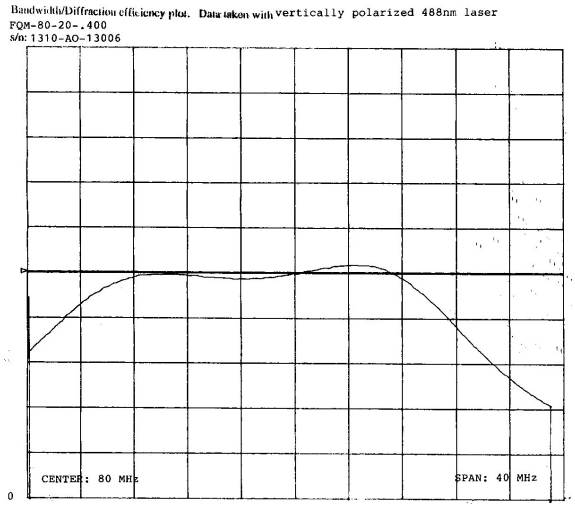


FIGURE 2.10 – Modulateurs Acousto-Optique

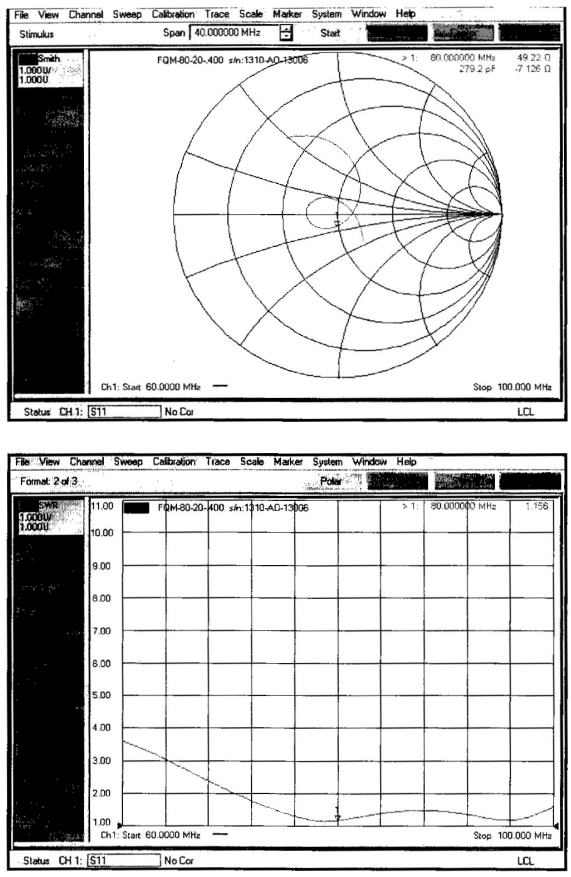


FIGURE 2.11 – Modulateurs Acousto-Optique

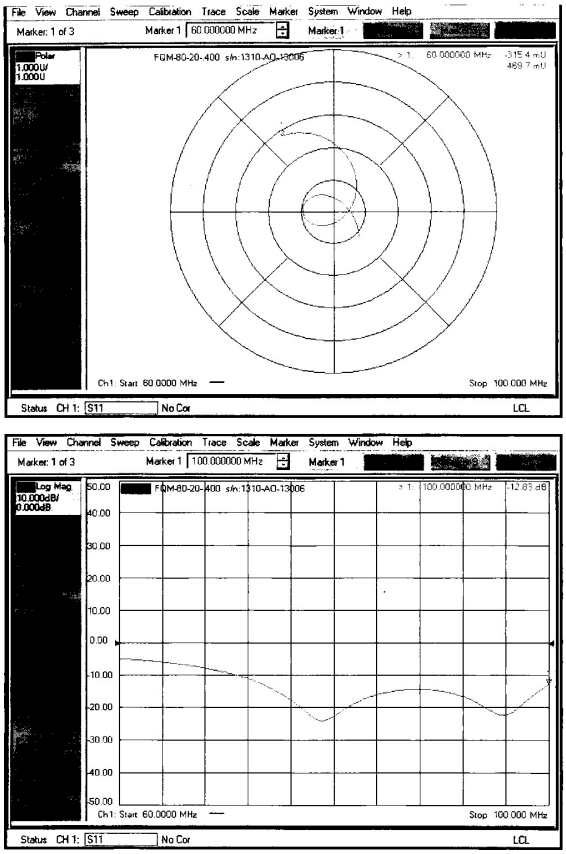


FIGURE 2.12 – Modulateurs Acousto-Optique

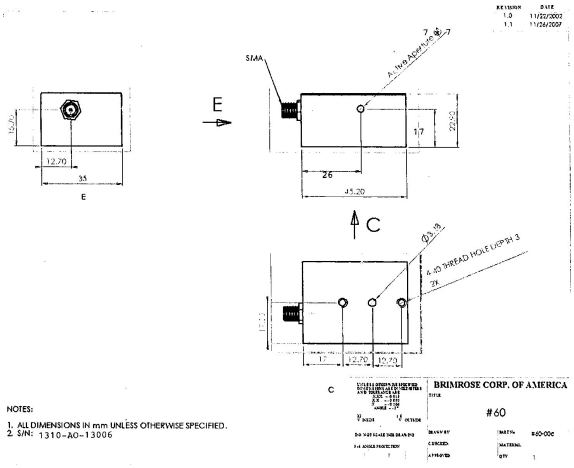


FIGURE 2.13 – Modulateurs Acousto-Optique

2.10 Racal Dana

L'appareil Racal Dana 1991 (figure 2.14) est utilisé comme compteur de photon. Sa configuration manuelle est difficile ; c'est pourquoi on préfère le configurer de façon logicielle. On établi donc la communication avec l'appareil à partir d'un ordinateur et on lui envoie les commandes appropriées.

Pour pouvoir bien fonctionner, le compteur de photon a besoin d'une référence temporelle. Pour ce faire, nous utilisons la sortie *10 MHz Output* comme entrée de *Input B*. Le comptage se fait proprement dit sur l'entrée *Input A*.

Cet appareil possède deux interfaces logicielles soient Matlab et Python.

FIGURE 2.14 – Racal Dana 1991

2.10.1 Matlab

2.10.2 Python

La classe RacalDana sert à interagir avec le compteur de photon Racal Dana 1991. Il s'agit d'une classe très ciblée puisqu'en dehors de l'initialisation on ne peut faire qu'une chose : mesurer. Voir Code 2.7 pour un exemple d'utilisation.

Code 2.7 – Exemple d'utilisation de *RacalDana avec Python*

```
import racalDana  # importe le module

# initialise la communication (on suppose l'adresse GPIB 3)
compteur = racalDana.RacalDana('GPIB0::3::INSTR')

# mesure pendant 5 secondes
comptes = compteur.measure(exposureTime=5)
```

2.11 SP275

Le spectromètre Acton Research SP275 [17] (Figure 2.15) est un spectromètre simple doté d'une tourelle de 3 réseaux (Table 2.3). Ce spectromètre fonctionne bien dans l'UV.

Pour contrôler ce spectromètre avec un ordinateur, on connecte celui-ci avec le contrôleur du spectromètre (SpectraDrive) par GPIB.

2.11.1 Matlab

La fonction SP275.m permet de contrôler le spectromètre avec Matlab. Code 2.8 donne un exemple d'utilisation simple.

17. Manuel : `http://ftp.piacton.com/Public/Manuals/Acton/SP-275_Obsolete.pdf`

TABLE 2.3 – Réseaux du spectromètre Acton Research SP275

Réseau	Densité (l/mm)	Blaze (nm)
1	2400	Hol
2	1200	300
3	1200	Hol

FIGURE 2.15 – Acton Research SP275

Code 2.8 – Exemple d'utilisation du SP275 avec Matlab

```
??? Code entre sans test. Verifier que c'est valide ???

SP275('initialise')  % initialise la communication

% Reseaux
SP275('?GRATINGS')  % liste des reseaux disponibles
SP275('?GRATING')  % lit le reseau actuel
SP275('GRATING', 3)  % selectionne un reseau en particulier

% decalage
SP275('OFFSET', 1)  %  defini le offset a 1 nm

% position
SP275('?NM')  % lit la longueur d'onde actuelle
SP275('GOTO', 532.1)  % deplace vers 532.1 nm
```

2.12 SR400

FIGURE 2.16 – Standford Research Systems SR400

2.12.1 Matlab

Code 2.9 donne un exemple d'utilisation simple.

Code 2.9 – Exemple d'utilisation du SR400 avec Matlab

```
??? Code entre sans test. Verifier que c'est valide ???
```

2.13 SR830

FIGURE 2.17 – Standford Research Systems SR830

2.13.1 Matlab

Code 2.10 donne un exemple d'utilisation simple.

Code 2.10 – Exemple d'utilisation du SR830 avec Matlab

```
??? Code entre sans test. Verifier que c'est valide ???
```

2.13.2 Python

Code 2.11 donne un exemple d'utilisation simple.

Code 2.11 – Exemple d'utilisation du SR830 avec Python

```
??? Code entre sans test. Verifier que c'est␣valide␣???
```

2.14 Trivista

Spectromètre triple [18] multi configurations.

FIGURE 2.18 – Princeton Instruments Trivista

18. ftp://ftp.princetoninstruments.com/public/Manuals/Princeton%20Instruments/TriVista%20System%20Manual.pdf

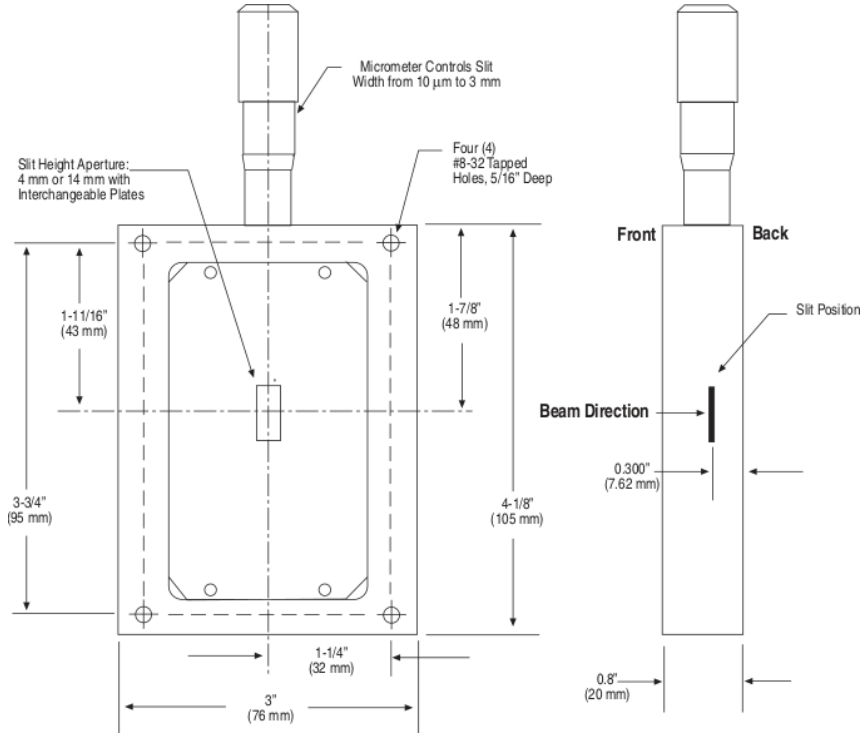


FIGURE 2.19 – Fente de sortie du Trivista. Le côté *Back* est l'extérieur du spectromètre.

2.14.1 Matlab

Code 2.12 donne un exemple d'utilisation simple.

Code 2.12 – Exemple d'utilisation du Trivista avec Matlab

```
??? Code entre sans test. Verifier que c'est valide ???
```

2.15 Verdi

2.16 Wavemeter

Le Wavemeter sert à donner une mesure précise de la longueur d'onde du laser sortant du Matisse.

2.16.1 Lecture

Il est possible de lire la longueur d'onde mesurée par l'instrument en utilisant deux faux ports série. Sur windows, il faut installer le logiciel *com0com* [19]. On crée ensuite deux ports séries (com 90 et com 91) reliés ensemble. Le logiciel du Wavemeter (qui doit être démarré et en mode acquisition) écrit ses résultats sur com 90 (???). Ce dernier recopie tout sur com 91. Il suffit de lire les résultats sur com 91. Un exemple de code est donné dans Code 2.13.

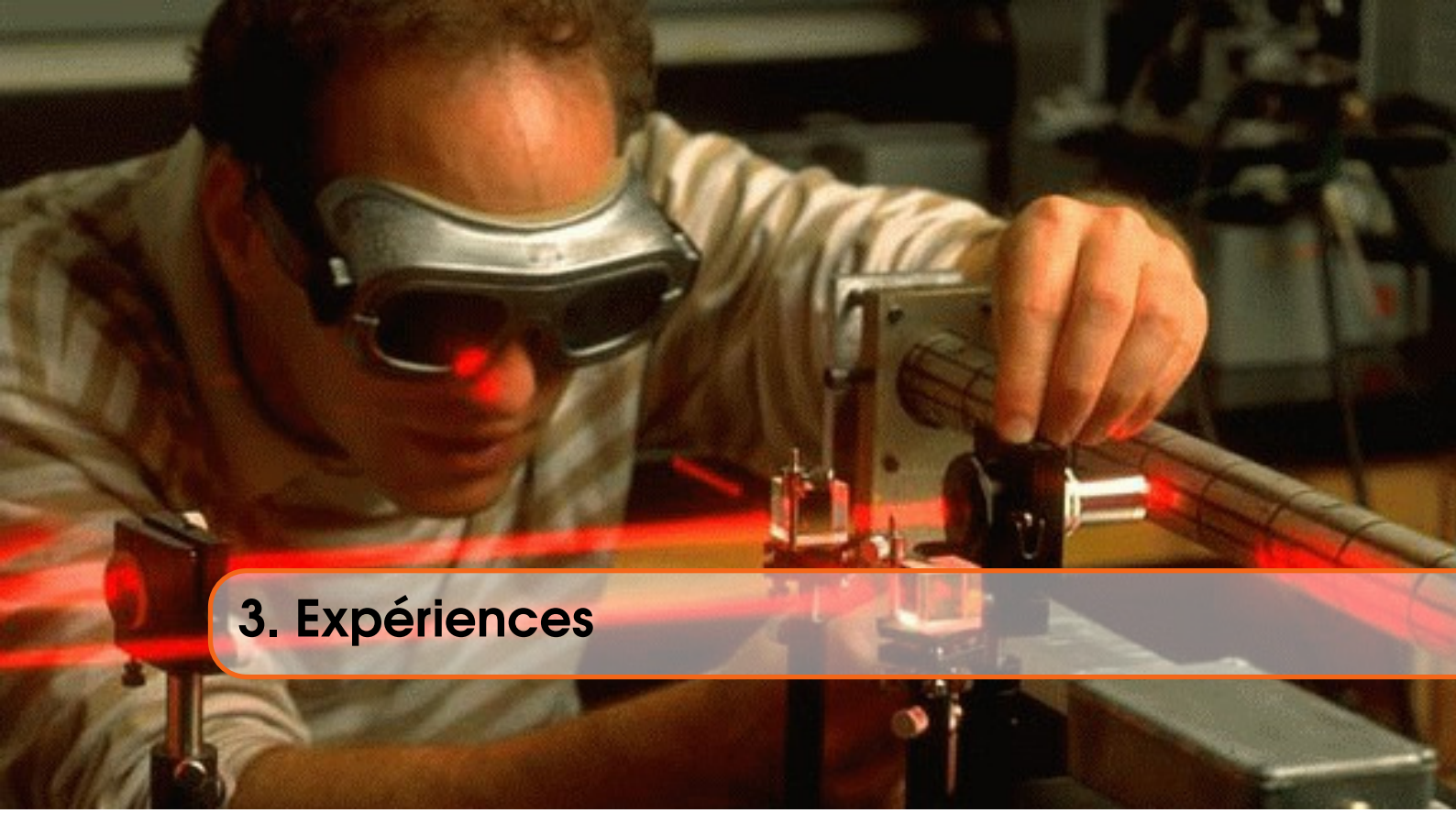
Cette fonction est implémentée dans la classe Matisse, dans la fonction (en fait une propriété) realWavelength.

19. `http://com0com.sourceforge.net`

2.16.2 Python

Code 2.13 – Exemple d'utilisation du Wavemeter avec Python

```
import Matisse
laser = Matisse.Matisse()

print(laser.realWavelength)
```



3. Expériences

Les expériences de photoluminescence résolue temporellement se font avec le module de Picoquant et le spectromètre Trivista. Dans Matlab, on utilise la fonction trivista_hydra.

trivista_hydra contrôle uniquement l'etage 3 du spectrometre. On suppose donc que le détecteur y est connecté. L'utilisateur est responsable de regler prealablement le spectrometre (mirroirs, fentes, reseau, etc). On peut par exemple utiliser tribalayage.

Voir Code 3.1 pour un exemple d'utilisation.

Code 3.1 – Exemples de photoluminescence résolue temporellement (PLRT) avec la Picoquant

```
trivista_hydra('l',[400:10:450], 't',30, 'dat', 'o',1, 'r',64,
    'f','output')
%Mesure la PLRT entre 400 et 450 nm par pas de 10 nm.
%Chaque accumulation dure 30 secondes.
%Cree des fichiers .dat (ascii) en plus des .mat.
%Utilise l'ordre 1 du spectrometre.
%Utilise une resolution de 64 ps.
%Sauve dans les fichiers output.mat et output.dat

trivista_hydra('l',[400:10:450], 't',2, 'b', 'f','output')
```

```
%Mesure la PL entre 400 et 450 nm par pas de 10 nm.
%Chaque accumulation dure 2 secondes.
%Sauve dans les fichiers output.mat
```

3.6 Raman

3.6.1 U1000

Pour faire une mesure de diffusion Raman sur le systeme U1000, il faut :

- Remplir le réservoir d'azote du détecteur CCD
- Winspec
  - Démarrer le programme *Winspec*
  - *Au besoin, vérifier que la caméra est détectée : Setup > Hardware ...*
  - *Vérifier la température du détecteur : Setup > Detector Temperature... > Target Temperature : -100 C, Current Temperature : Locked*
- Déterminer visuellement la position du spectrometre U1000 (compteur sur le côté)
- Interface Python
  - Démarrer le programme *Spyder*

Les mesures se font en utilisant la Console dans Spyder. Note importante : toutes les unités sont en Angtroms ou en cm-1 (relatifs).

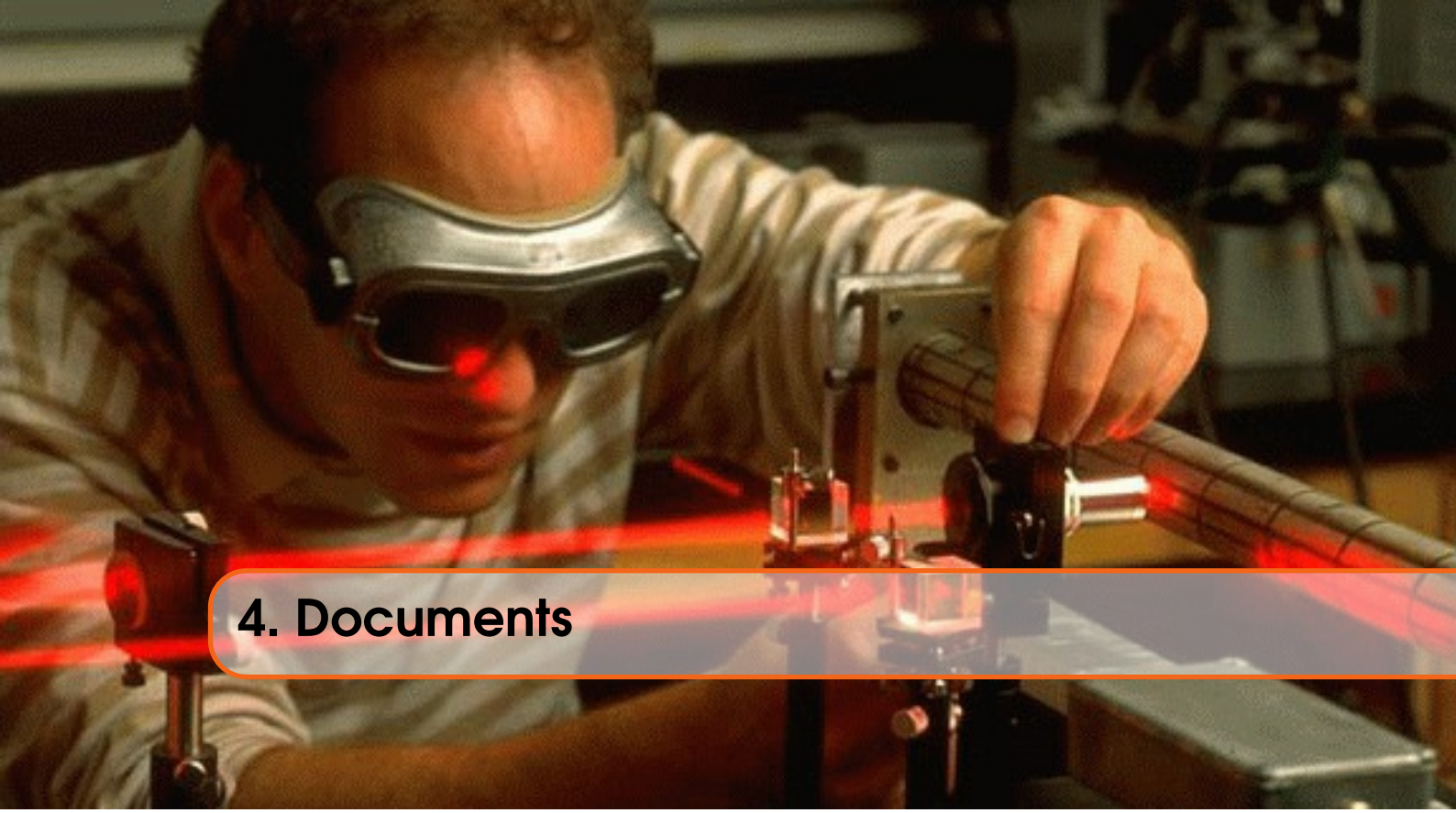
Code 3.2 – Mesure Raman avec le U1000 et sa CCD

```
# ------- Repertoire de travail -------
#
cd 'C:\\Documents and Settings\\Admin\\Sync\\Python'

# ------- Importation du module necessaire -------
#
import test_mesure as mesure


# ------- Initialisation -------
#
# Initialisation de l'objet representant l'experience
#
# Note importante:
# On doit proceder a cette initialisation une seule fois. S'il y
    a un probleme de communication, le plus simple est de fermer
    Spyder et recommencer (en attendant que des instructions a ce
    propos soient ecrites).
#
# lire visuellement la longueur d'onde sur le compteur du
    spectrometre
m = mesure.Mesure(5399)  # ici le spectrometre se trouve a 5399
    Angstroms
# m represente l'instrument de mesure
# C'est en utilisant ses fonctions et methodes qu'on fait les
    mesures, deplace le spectrometre, etc.
#
# Aide et instructions
help(m)  # Objet general representant la mesure. Fonctions
    disponibles, etc.

# ------- Longueur d'onde du laser (necessaire pour mesure Raman)
    -------
```

```
#
m.setLaser(5321)  # ici le laser est a 5321 Angstroms


# ------- Position du spectrometre -------
#
# Lire
m.A()  # en A
m.wavenumber()  # en cm-1 relatif
#
# Deplacer
m.A(position_cible_en_A)
# ou
m.wavenumber(position_cible_en_cm-1_relatif)


# ------- Decalage du spectrometre -------
#
# Au besoin, entrer le decalage du spectrometre entre affichage
    et realite
#
# Etape 1
m.spectrometer.positionOffset = 0  # on met le decalage a nul
    pour pouvoir le determiner
#
# Etape 2
# Faire une mesure sur un pic de longueur d'onde connue (voir par
    exemple plus bas pour la raie du Hg)
# Pour determiner la position mesuree, on peut deplacer le
    spectrometre (voir plus haut) jusqu'a ce que le pic soit au
    centre de la CCD (en visualisant dans Winspec le pixel 670).
# Par exemple, ici lorsqu'on entre
#     m.A(5458.97)
# on constate (dans Winspec) que le pic est centree sur le pixel
    670
positionVraie = 5460.735  # Valeur tabulee
positionMesure = 5458.97  # Valeur mesuree par la procedure de
m.spectrometer.positionOffset = positionVraie - positionMesure


# ------- Fonction de mesure -------
#
# La fonction measureRange sert a faire l'acquisition
#
# Aide et instructions
help(m.measureRange)  # Mesure en tant que tel
# def measureRange(self, Range , nOverlap=3, accTime=1, images=
    False, unit='cm-1', maskCCD=None, plot=True, spectroSlits=10,
    detector='CCD', rootFilename='.', baseFilename=None, sample
```

```
    ='', comment=''):
#         """
#         Measure in a range of position
#         Each point is garanted to come from the have the same
    number of points
#
#         Range
#             Range of the of measurement
#             [Begin, End]
#             End is optionnal. If omited, will take one window
    centered on Start
#         nOverlap
#             number of time the same position will be measured
#             (the CCD detector will be centered at different
    positions)
#             (integer)
#         accTime
#             accumulation time per position per image (second)
#             (float)
#         images
#             number of measurement from the same detector
    centered position
#             If False (default) automaticaly choose the best
    setting
#             (integer)
#         unit
#             unit of position
#                 'A'
#                 'cm-1' (default)
#         maskCCD
#             valid index on the CCD detector
#                 [firstValidIndex, lastValidIndex] (integers)
#                 ex.: [175, 1125] (default)
#         spectroSlits
#             width (mm) of the intermediary spectrometer slits
#                 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
#             This determine the camera maskCCD
#             (integer)
#         detector
#             Detector name
#                 'CCD'
#                 'PMT'
#             (string)
#         rootFilename
#             directory where to save the data
#             (string)
#         baseFilename
#             leading part of filename (will be numbered)
#                 None: get current date in format AAMMDD
```

```
#             (string)
#         """


# ------- Mesure Raman -------
#
# (Exemple)
#
# pour faire une mesure entre 150 et 550 cm-1
# accTime: Acquisition de 10 secondes par position
# images: 5 repetitions par position (a partir de 5, un
    algorithme enleve efficacement les pics du aux "rayons
    cosmiques"
# nOverlap: Chaque position est mesuree a 4 endroits differents
    sur la CCD (pour enlever le flatfield). Autrement dit, chaque
    mesure successive se deplace de 1/4 de la plage totale de la
    CCD, donc chaque position est mesuree 4 fois.
# mesure en cm-1 relatif
x,y = m.measureRange([150, 550], unit='cm-1', accTime=10, images
    =5, nOverlap=4)


# ------- Mesure Photoluminescence, etc -------
#
# (Exemple)
#
# pour mesurer la raie du Hg
# mesure en Angstroem
# Ici la plage est un nombre seul (5461). Ce sera la position
    centrale de la CCD.
x,y = m.measureRange(5461, accTime=1, images=10, unit='A')
#
# mesure entre 6500 et 6700 A
# mesure en Angstroem
x,y = m.measureRange([6500, 6700], unit='A'), accTime=1, images
    =10, nOverlap=5)
```



4. Documents

Feuille d'utilisation Laboratoire de R. Leonelli

Coordonnées

Utilisateur	Responsable
Nom :	Nom :
Téléphone :	Téléphone :
Courriel :	Courriel :

Expérience

Mesure	Excitation	Spectrometre	Détection	Cryostat
Absorption	Lampe blanche	U1000	PD Si	Oxford (Bain)
PL	Laser He-Ne	Bomem	PD InGaAs	Cryo (Plat)
PLE	Millenia	SP275	PD InSb	Cycle fermé
Raman	Matisse	Trivista	CCD (U1000)	(gaz échange)
TCSPC	Wavetrain		PMT R943-02	(doigt froid)
Photoréflectance	Verdi		Pixis	
	Mira		PDA InGaAs	
	Doubleur Mira		SPAD	

Hélium liquide :

Description de l'expérience

Horaire d'utilisation

Date	Heure début	Heure fin	Durée (heure)

Remarques au verso. En particulier, signaler tout bris d'équipement

Index