Institut d'Optique Graduate School
Interfaçage Numérique

INTERFAÇAGE NUMÉRIQUE

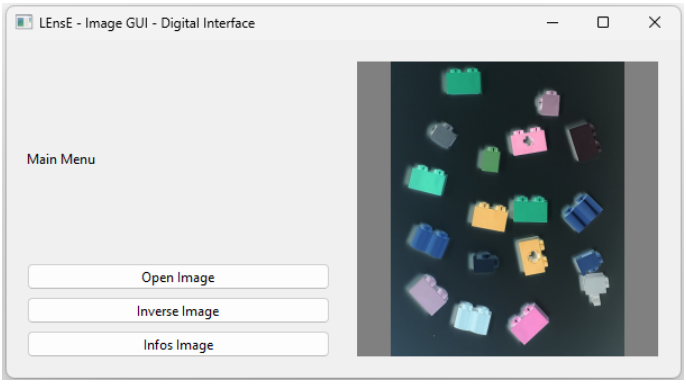
Travaux Pratiques

Semestre 6

Développement d'une interface en PyQt6

---

2 séances



Ce sujet est disponible au format électronique sur le site du LEnsE - https ://lense.institutoptique.fr/ dans la rubrique Année / Première Année / Interfaçage Numérique S6 / Bloc 2 Caméra, Images et Interfaces / Interface Humain-Machine.

Développement d'une Interface en PyQt6

À l'issue des séances de TP concernant le **bloc de développement d'une interface**, les étudiant·es seront capables de **développer une interface Humain-Machine simple** à partir d'un exemple pour ouvrir une image et afficher le résultat d'un traitement sous forme d'un graphique.

Pour cela, ils·elles seront capables de :
— **ajouter des éléments graphiques** à une interface existante ;
— **gérer les événements** liés aux éléments graphiques dynamiques (boutons...) ;
— **afficher un graphique ou une image** dans un conteneur graphique de l'interface.

Objectifs du bloc

L'objectif principal de ce bloc est de **réaliser une interface humain-machine permettant d'afficher le résultat d'une simulation simple** ou de données.

Cette interface sera codée à l'aide de la bibliothèque **PyQt6** et les étudiant·es seront amené·es à découvrir les éléments de base d'une telle interface.

Cette séquence de TP est basée sur le langage **Python**. Vous pourrez utiliser l'environnement **PyCharm** (édition Community 2023 - ou supérieure) et **Python 3.10** (ou version supérieure - inclus dans la distribution Anaconda 3).

Ressources

Un **tutoriel sur PyQt6** est disponible à l'adresse suivante :
https://iogs-lense-training.github.io/python-pyqt-gui/
Des **codes d'exemple** associés à ce sujet sont disponibles à l'adresse suivante :
https://lense.institutoptique.fr/ressources/Annee1/InterfacageNumerique/bloc_gui/codes/

Il existe également une multitude de tutoriaux et d'exemples sur Internet d'application basée sur **PyQt6**. Par exemple : https://www.pythonguis.com/pyqt6-tutorial/.

Enfin, n'hésitez pas à vous rendre sur la **documentation officielle** de Qt pour Python :
https://doc.qt.io/qtforpython-6/

Déroulement du bloc

Ce bloc se déroule en deux temps :

— une première partie où vous serez guidé·es pour **découvrir les éléments de base d'une interface** développée à l'aide de la bibliothèque **PyQt6**, incluant une gestion des événements ;

— une seconde partie où vous serez plus en autonomie pour **répondre à l'un des sujets proposés** autour de l'**affichage de données de simulation** ou tout autre traitement de données.

Premiers éléments graphiques et interactions

Etape 1 - 30 min Etudier la structure de base d'une application PyQt6 (sans évènement)

Etape 2 - 30 min Intégrer de nouveaux éléments graphiques

Etape 3 - 60 min Utiliser des signaux pour gérer des évènements

Etape 4 - 60 min Ouvrir une image et l'afficher

Etape 5 - 60 min Afficher un graphique

Affichage de données de simulation ou de traitement

Le but final est d'afficher des données provenant soit d'un fichier à ouvrir : histogramme d'une image, données dans un fichier CSV... , soit d'une simulation/observation/modélisation : réponse à un échelon, ...

Premiers éléments graphiques et interactions

Objectifs de la séance

Dans cette première partie, vous allez **étudier des exemples de code** permettant d'afficher certains éléments graphiques dans une IHM **PyQt6** et de réaliser des actions associées à certains de ces éléments.

Installation de PyQt6

La bibliothèque **PyQt6** n'est pas installé par défaut dans la plupart des distributions. Vous devrez l'installer par l'intermédiaire de la commande suivante :

```
1 pip install pyqt6
```

Selon les distributions, vous devrez exécuter cette instruction soit dans le terminal (ou Invite de commande sous Windows) soit à l'aide du Prompt d'Anaconda.

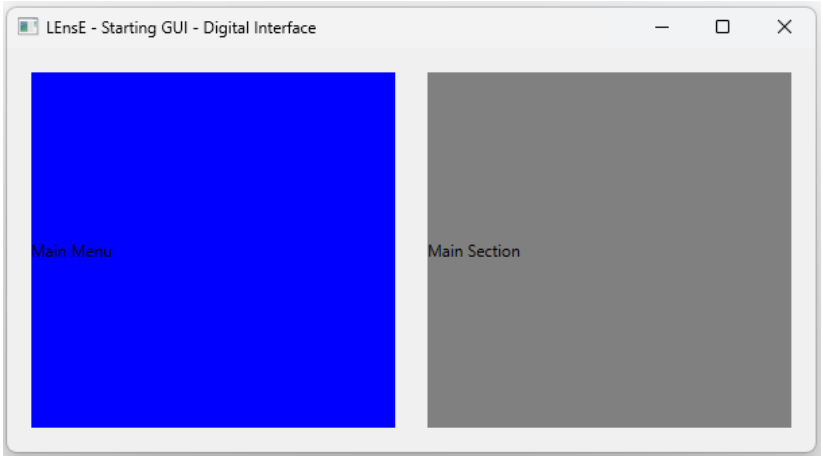
Etudier la structure de base d'une application PyQt6

Temps conseillé : 30 min

Dans cette section, vous allez étudier une application simple basée sur PyQt6 afin de **découvrir les briques élémentaires** à mettre en oeuvre dans ce type d'interface.

A partir du fichier **start_gui.py** :

— Lancer l'application et visualiser le résultat.
— Étudier un peu plus en détail la structure du code fourni et notamment les différents types de conteneurs utilisés. Faire un schéma des différents éléments et leurs liens.



Intégrer de nouveaux éléments graphiques

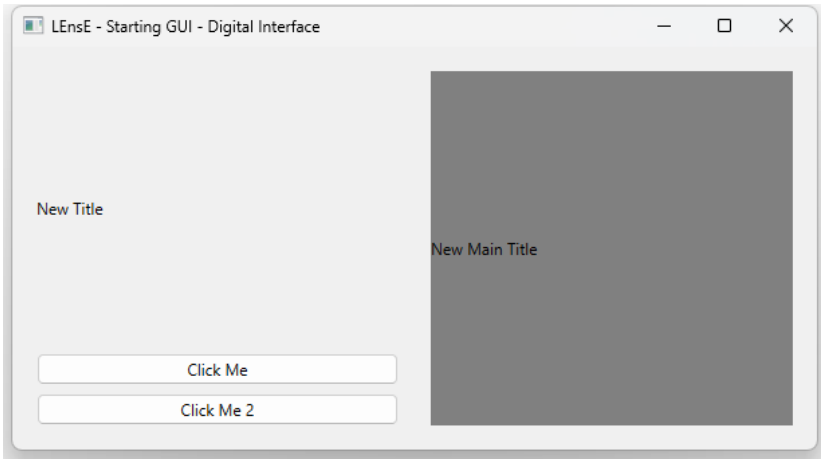
Temps conseillé : 30 min

L'interface précédente n'inclut pas d'objets graphiques interactifs (bouton, zone de sélection...). Nous allons dans cette section **ajouter quelques boutons**.

A partir du fichier **start_gui.py** :

— Ajouter un bouton (**QPushButton**) nommé ***first_button*** au menu principal
— Ajouter une action à ce bouton qui appelle la fonction ***action_clicked*** lorsqu'on clique dessus
— Ajouter un second bouton nommé ***second_button*** au menu principal, qui appelle la fonction ***action_clicked*** lorsqu'on clique dessus.
— Tester votre application.

— Modifier les actions associées selon le bouton appuyé :
  — ***first_button*** : renomme le nom du menu principal ;
  — ***second_button*** : renomme le nom de la fenêtre principale.



Utiliser des signaux pour gérer des évènements

Temps conseillé : 60 min

Avec la structure précédente, il est nécessaire que les éléments graphiques contenus dans un **QWidget**, par exemple, connaissent l'élément graphique dit parent afin de pouvoir interagir avec celui-ci.

Une autre méthode consiste à utiliser des signaux (objet de la classe **pyQtSignal**) pour transmettre des informations d'un objet à un autre. L'exemple suivant propose d'étudier cette méthode.

A partir du fichier **signal_gui.py**

— Lancer l'application et visualiser le résultat.
— Étudier la nouvelle structure et trouver les points communs et les différences avec la précédente structure de code.

— Ajouter une fonction ***set_title*** à la classe **MainWidget** pour modifier le titre de la section principale
— Modifier l'application pour obtenir les mêmes interactions que précédemment avec les deux boutons, mais sans modifier la fonction ***action_clicked*** de la classe **MainMenuWidget**

Ouvrir une image et l'afficher

Temps conseillé : 30 min

Il est parfois utile que l'utilisateur$cdot$trice puisse ouvrir un fichier. Nous allons ici nous intéresser à l'ouverture et l'affichage d'un fichier contenant une image (type *jpg*).

A partir du fichier **image_gui.py**

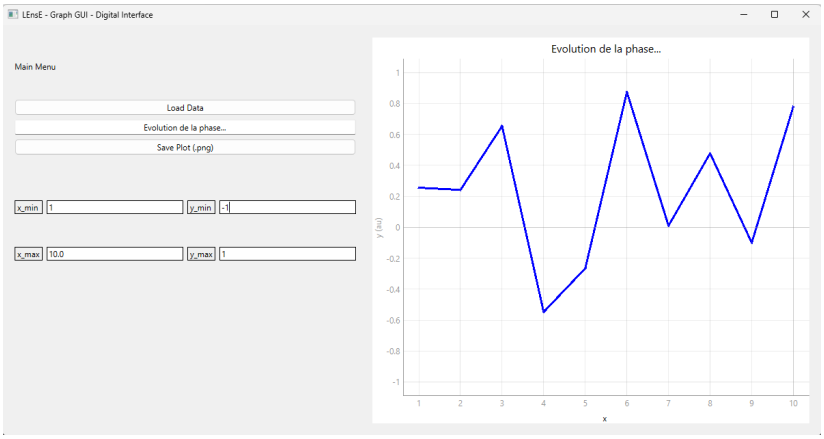
— Lancer l'application et visualiser le résultat.
— Ouvrir une image à l'aide de l'interface.
— Étudier la structure de l'application proposée, notamment les liens entre les différentes classes.

— Ajouter un bouton dans le menu principal permettant d'inverser l'affichage des couleurs de l'image et modifier le code pour que l'affichage soit mis à jour.
— Ajouter un bouton dans le menu principal permettant d'afficher dans la console la taille de l'image et la moyenne de l'ensemble de ses pixels.
— Modifier le code de l'application pour que les boutons d'inversion et d'affichage des informations de l'image soit par défaut non utilisable et qu'ils le deviennent uniquement lorsqu'une image est chargée.
— BONUS : Modifier le comportement du bouton d'affichage des informations pour qu'il affiche en plus l'histogramme (pour les 3 couleurs de l'image) dans une fenêtre Matplotlib classique (hors interface pyQt6).

Afficher un graphique

Temps conseillé : 60 min

Dans cette section, nous allons nous intéresser à l'affichage et la mise à jour d'un graphique à l'intérieur d'un conteneur graphique.



A partir du fichier **graph_gui.py**

— Lancer l'application et visualiser le résultat en chargeant les données du fichier **data.csv** fourni
— Modifier la couleur du fond du graphique ainsi que l'épaisseur et la couleur d'affichage de la courbe.
— Désactiver l'utilisation du bouton *Save Plot (.png)* par défaut et autoriser la sauvegarde lorsque des données sont présentes.

On se propose maintenant de pouvoir **interagir avec le graphique** en permettant d'en changer certains paramètres (limites d'affichage en X et en Y).

Une classe **RangeWidget**, qu'il faudra compléter, a été conçue à cet effet. Des zones de texte ont été prévues pour saisir les limites d'affichage en X et en Y du graphique. Ces zones ne sont pas éditables tant que des données n'ont pas été chargées.

— Ajouter un objet de type **RangeWidget** dans le menu principal.
— Modifier le code pour activer les zones de texte (x_min, $x\,\max$...) lors du chargement des données.
— Ajouter une méthode ***set_lim*** à la classe **RangeWidget** permettant de mettre à jour les limites des axes du graphique. *Les zones de texte fournissent des chaines de caractères et non des nombres...*
— Modifier le code pour que la fonction ***set_lim*** soit appelée à chaque fois que les zones de texte sont modifiées par l'utilisateur$cdot$trice. *Indice : utilisation de la méthode* ***setRange*** *de* ***pyQtGraph*** *et du signal* ***textChanged*** *de* ***QLineEdit****.*
— Ajouter une zone de texte (**QLineEdit**) dans le menu principal permettant de modifier le titre du graphique.
— Modifier le code pour que la sauvegarde du graphique sous forme d'une image PNG soit faite dans un fichier qui porte le titre du graphique.

Affichage de données de simulation

Vous devrez traiter l'un des sujets suivants, **au choix**.

Sujet A : Filtrage par TF

A partir du fichier **image_filter_gui.py**, on souhaite pouvoir :
— ouvrir une image,
— calculer la TF de cette image (FFT2D),
— créer un masque sur la TF (circulaire par exemple ou rectangulaire),
— générer l'image résultante (par TF inverse) et l'afficher.
Ce sujet peut-être mis en lien avec le TP de filtrage en Optique S6.

Sujet B : Masque sur une image

A partir du fichier **draw_mask_gui.py**, on souhaite pouvoir :
— créer un masque rectangulaire à partir de deux points sélectionnés à la souris,
— stocker le masque dans une matrice de la même taille que l'image,
— afficher l'image avec le masque dans une nouvelle fenêtre,
— calculer l'histogramme des deux images et les afficher (pour les comparer).

Il est possible d'accélérer le traitement des calculs par l'utilisation d'une matrice *Numpy* qui ne prend pas en compte les valeurs qui ne sont pas incluses dans le masque :

```
1  new_image = np.ma.masked_where(np.logical_not(mask), image)
```

L'objet *mask* doit contenir des données de type booléen.

Sujet C : Coupe dans une image

A partir du fichier **image_slicer_gui.py**, on souhaite pouvoir :
— ouvrir une image,
— spécifier une ligne (ou une colonne particulière) selon laquelle on souhaite avoir une coupe de l'image,
— afficher la coupe de l'image selon la ligne (ou colonne) sélectionnée.
Cette méthode est utilisée pour analyser des images ayant un profil particulier : faisceau laser (projet en ONIP-1), tâche de diffraction (TP d'Optique S6 - comparaison avec une fonction de Bessel)...