1. Introducción

El programa para controlar el SLM se creó utilizando el sistema de *módulos* del lenguaje *Python*. El módulo incluye una varidad de funciones que permiten enviar imágenes al SLM de modo que sean utilizadas como hologramas. El código está escrito para funcionar en **Ubuntu**.
La idea principal es abrir una imagen en el programa Eye Of Gnome (EOG) y editar esta imagen para que sea el holograma que se quiere proyectar en el SLM. El programa EOG desplegará entonces imágenes en pantalla completa en el 'monitor' (que en realidad es el SLM), de modo que el usuario no tiene acceso visual a estas imágenes.
El módulo contiene cuatro archivos y una carpeta con imágenes para corregir aberraciones generadas por el reflejo de la luz en el SLM. A continuación se enlistan dichos archivos con una pequeña descripción:

- src.py *: archivo con código en Python. Es la base del programa.*
- dospi.py *: archivo que contiene información sobre la normalización de la escala de grises.*
- imagen.png *: archivo imagen que se mandará al SLM.*
- ManualModuloPython.pdf *: es este manual.*
- PatronCorrector/
  - n2-910nm.bmp *: archivo imagen para corregir aberraciones a 910 nm.*
  - n3-780nm.bmp *: archivo imagen para corregir aberraciones a 780 nm.*
  - n4-633nm.bmp *: archivo imagen para corregir aberraciones a 633 nm.*

El módulo se puede descargar en https://github.com/shgaeo/MisPys/tree/master/modulador.

2. Configuración previa

En Python, para poder cargar módulos sin estar en la carpeta que los contiene es necesario agregar al archivo `/HOME/.bashrc` la siguiente línea:

```
export PYTHONPATH="${PYTHONPATH}:/HOME/Documentos/MisPys"
```

Siendo `/HOME/Documentos/MisPys` la carpeta que contiene los módulos (por ejemplo, en mi caso es: `/home/santiago/Documentos/MisPys`).

El programa tiene dos opciones: una en que se utiliza el SLM como segundo monitor y otra en que el SLM es el único monitor conectado a un CPU que controlamos de manera remota utilizando ssh. Ambas opciones se pueden elegir dentro del programa pero cada una requiere una pre-configuración del CPU distinta. A continuación se explica como realizar ambas configuraciones.

2.1. SLM como segundo monitor

La idea en este caso es que el SLM sea un segundo monitor cuya única función sea desplegar una imagen en 'fullscreen'. Primero que nada es necesario que el SLM aparezca como segundo monitor y a la derecha del monitor principal. Si no es este el caso, se debe abrir `Configuración del sistema`, elegir la opción `monitores` y mover con el `mouse` la posición del segundo monitor de modo que esté a la derecha del principal.
Por otro lado es necesario que se instale la paquetería wmctrl, que permite modificar la posición de una ventana abierta en nuestro escritorio. La instalación se lleva a cabo con el comando :

```
shell> sudo apt-get install wmctrl
```

¡LISTO! Ahora puedes comenzar a utilizar tu SLM.

2.2. SLM como único monitor (acceso via ssh)

La idea en este caso es controlar de manera remota (vía ssh) el CPU donde se encuentre instalado el programa de control, el cual despliega imágenes en pantalla completa en su único monitor, que en este caso es el SLM. A este tipo de CPU (sin acceso a monitor) se le conoce como *headless* CPU.
Antes de conectar el *headless* CPU al SLM es necesario configurarlo, para lo cual es necesario conectarle un monitor real.
Una vez hecho esto es necesario eliminar la necesidad de iniciar sesión al arranque del CPU. Para lo cual es necesario abrir `Configuración del sistema`, elegir la opción `Cuentas de usuario` y seleccionar el botón `Iniciar sesión automáticamente`.
Además es necesario configurar este CPU como servidor, de modo que es necesario instalar ssh server con el siguiente comando:

```
shell> sudo apt install openssh-server
```

NOTA: En el CPU que vayas a usar para conectarte a este servidor debes instalar `openssh-client` en lugar de `openssh-server`.
Para poder conectarte a este servidor necesitas conocer tres cosas:

1. Nombre de usuario: Para conocerlo solo abre una terminal (`ctrl+t`) y el nombre de usuario debe ser lo que aparece antes del símbolo @.
2. Dirección IP: Para conocer tu dirección IP puedes usar el comando `ifconfig`.
3. Contraseña: Es la contraseña asociada a este usuario.

Apunta bien estos datos, ya que una vez que el CPU sea *headless* ya no podrás acceder a los mismos.

¡LISTO! Ahora puedes desconectar todo de tu CPU y conectar únicamente la alimentación y el controlador del SLM vía conexión DVI. Y no olvides el cable de red para poder tener acceso vía ssh.

3. Lista de funciónes

A continuación se enlistan las funciones disponibles tras cargar el módulo. Nótese que al cargar el módulo no solo se obtiene acceso a las funciones, también se define el valor de la constante `CONST_2PI`, este valor representa el tono de gris que corresponde a la fase 2π y depende de la longitud de onda del láser con que se trabaje, de modo que previo a cargar el módulo se tiene que verificar que este valor sea el adecuado.

- `inicia()`
  Esta función debe correrse al inicio de cada sesión si se va a trabajar con el SLM. Lo que hace es: abrir el archivo `imagen` en modo pantalla completa. Nótese que el archivo `imagen` es una imagen blanca de 800×600 pixeles y que se abrirá con el programa *eog* (instalado por default en *Ubuntu*).
- `finaliza()`
  Esta función debe correrse al final de cada sesión en la que se haya utilizado la función `inicia`, ya que cierra el programa *eog*.
- `monitor2(imagen:‘‘Image’’)`
  Esta función re-escribe el archivo `imagen` de modo que cambia lo que se proyecta en el SLM. El argumento de la función debe ser una imagen de 800×6000.
- `grayImage(matInt:‘‘ndarray’’)`
  Esta función toma como argumento una matriz de enteros y regresa una imagen en escala de grises de 8-bits.
- `grayImageCorr(matInt:‘‘ndarray’’)`
  Esta función hace lo mismo que la función `grayImage` pero suma a la matriz original el patrón de corrección de aberraciones.

- `escalon(periodo,dosPi=CONST_2PI,fondo=0,cols=800,rengs=600)`
  Esta función genera una rejilla binaria en forma de matriz de enteros de `cols`×`rengs`. La función tiene control sobre el periodo y profundidad de la rejilla con los primeros tres argumentos. Si no se especifican, la altura será la constante `CONSR_2PI`, el fondo será cero y el tamaño será 800×600.

- `blazeMat(periodo,dosPi=CONST_2PI,cols=800,rengs=600)`
  Esta función genera una rejilla de diente de sierra en forma de matriz de enteros de `cols`×`rengs`. La función tiene control sobre el periodo y profundidad de la rejilla con los primeros dos argumentos. Si no se especifican, la altura será la constante `CONSR_2PI` y el tamaño será 800×600

- `thetaMat(th=0)`
  Esta función genera una matriz cuyas entradas representan los ángulos (van de $-\pi$ a π) correspondientes a la posición de las entradas de la matriz. El argumento `th` corresponde a los grados por los cuales se puede rotar el holograma. Si no se especifica se tomará `th`$= 0$.

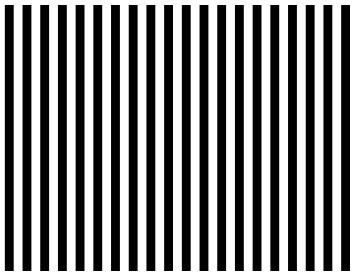
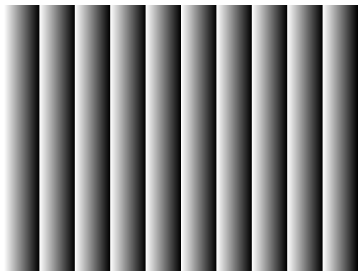
- `thetaMatInt(n,dosPi=CONST_2PI,th=0)`
  Esta función toma la matriz de la función `thetaMat(th)` y la convierte en enteros de 8-bits, donde el valor máximo del arreglo es el valor de la variable `dosPi`. Además multiplica la matriz de enteros resultante por `n`, haciendo así que el holograma helicoidal tenga `n` vueltas. Esta es la función que se utiliza para inducir la fase tipo vórtice al haz para (junto con un axicón) generar un haz Bessel de orden `n`. Si no se especifican, el ángulo `th` será cero y el valor máximo de arreglo será `CONST_2PI`.

- `calibrar()`
  Esta función es para calibrar la relación *tono de gris − fase inducida* para una longitud de onda del láser. Lo que hace es proyectar en el SLM un holograma tipo rejilla binaria variando su profundidad de 0 a 255. Para calibrar se debe colocar una lente que enfoque el haz reflejado en el modulador y realizar una fotografía del haz en dicho foco (hasta ahora esto debe hacerse manualmente y de manera independiente) para cada valor de la profundidad de la rejilla.

Antes de mandar dichas imágenes es necesario utilizar la función `inicia()` que despliega una imagen blanca en el SLM, la cual puede ser reemplazada posteriormente por los hologramas deseados. Esto lo hace utilizando comandos de *Linux*. Una vez inicializado el programa se pueden utilizar las funciones listadas anteriormente para generar los hologramas deseados y mandarlos al SLM utilizando la función `monitor2(imagen:‘‘Image’’)`, la cual toma como argumento una imagen y la despliega en el SLM. Al terminar de utlizar el programa se debe usar la función `finaliza()` que cierra el programa *eog*.

4. Ejemplos

A continuación se muestra una tabla con ejemplos de algunas funciones utilizadas en el módulo:

In[]	grayImage(escalon(40))	grayImage(blazeMat(80))	grayImage(thetaMatInt(1))
Out[]			

Cuadro 1: Muestra de lo que hacen algunas funciones cargadas en el módulo. Si estas imágenes se utilizan como argumento en la función `monitor2` entonces se mandarán para proyectarse como hologramas en el SLM.

Finalmente se muestra un ejemplo de los comandos utilizados para: conectarse vía ssh al CPU conectado al SLM, abrir *Python* y al mismo tiempo permitir que se desplieguen imágenes en el CPU remoto, cargar el módulo, iniciar la comunicación con el controlador del SLM, mandar vórtices de distintos órdenes al SLM, finalizar comunicación y cerrar *Python*. Nótese que para que esto funcione se debe haber realizado la configuración previa (sección2) y debe estar conectado el controlador del SLM vía DVI.

```
shell-client> ssh controlslm@10.10.6.178
...
shell-server>  export DISPLAY=:0 && python
...
>>> from modulador.src import *
>>> inicia()
>>> monitor2(grayImage(thetaMatInt(0)))
>>> monitor2(grayImageCorr(thetaMatInt(0)))
>>> monitor2(grayImage(thetaMatInt(1)))
>>> monitor2(grayImageCorr(thetaMatInt(1)))
>>> monitor2(grayImage(thetaMatInt(4)))
>>> monitor2(grayImageCorr(thetaMatInt(4)))
>>> finaliza()
>>> exit()
...
shell-server>
```

NOTA: En el servidor remoto, los siguientes comandos sirven respectivamente para apagar o para reiniciar el CPU:

```
shell-server> sudo poweroff
shell-server> sudo reboot
```