

1. Introducción

El programa para controlar el SLM se creó utilizando el sistema de *módulos* del lenguaje *Julia*¹. Se creó un módulo que incluye una variedad de funciones que permiten enviar imágenes al SLM de modo que sean utilizadas como hologramas. El código está escrito para funcionar en **Ubuntu**.

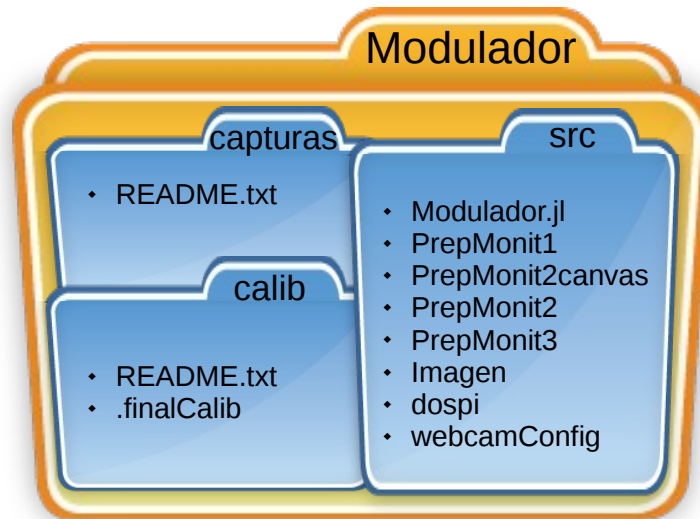


Figura 1: Archivos y carpetas incluidos en el módulo

En la figura 1 se muestra el contenido del módulo. En la carpeta **calib** se guardan temporalmente imágenes durante el proceso de calibración del modulador en función de la longitud de onda, esta información se guarda y se lee en el archivo **.finalCalib**. En la carpeta **src** se encuentra el archivo **Modulador.jl** que es el corazón del módulo, los demás elementos de la carpeta se explicarán más adelante (ver sección 3).

2. Configuración previa

NOTA: En lo siguiente utilizaré como directorio home **/home/santiago/**, quien use esto debe utilizar su propio directorio home.

Para poder cargar el módulo con el comando

```
julia> using Modulador
```

es necesario crear un archivo **/home/santiago/.juliarc.jl** que contenga la siguiente línea:

```
push!(LOAD_PATH, "/home/santiago/Documentos/MisModulos")
```

Donde el texto entre comillas es el path donde está el módulo deseado.

Además el módulo utiliza algunos programas de *Ubuntu* que es necesario instalar por separado. A continuación se incluyen estos programas:

- **Compiz configuration settings manager (ccsm)**

Para instalar este programa abrir la ventana de comandos (**ctrl+t**) y escribir

```
shell> sudo apt-get install compizconfig-settings-manager
```

¹<http://julialang.org/>

Este programa permite personalizar el escritorio de Ubuntu. En el caso del módulo lo que se requiere es que al abrir la aplicación *Eye of Gnome* (eog) se despliegue en la posición del segundo monitor. Para cargar esta configuración es necesario seguir los pasos siguientes (después de haber instalado ccsn):

Primero correr los siguientes comandos:

```
|+shell>+| cd
|+shell>+| mkdir .dconforg
|+shell>+| dconf dump /org/ > $\sim$/.dconforg/dconforg.dat
|+shell>+| cp $\sim$/.dconforg/dconforg.dat
$\sim$/.dconforg/dconforg-2ndscreen-EOG.dat
|+shell>+| gedit ~/.dconforg/dconforg.dat
```

Primero correr los siguientes comandos:

```
shell> cd
shell> mkdir .dconforg
shell> dconf dump /org/ > /home/santiago/.dconforg/dconforg.dat
shell> cp /home/santiago/.dconforg/dconforg.dat
/home/santiago/.dconforg/dconforg-2ndscreen-EOG.dat
shell> gedit /home/santiago/.dconforg/dconforg.dat
```

Agregar el final del archivo las siguientes líneas:

```
[compiz/profiles/unity/plugins/place]
position-x-values=[0]
position-y-values=[0]
position-matches=['name=eog']
position-constrain-workarea=[true]
```

Guardar y cerrar. Correr comando:

```
shell> gedit /home/santiago/.dconforg/dconforg-2ndscreen-EOG.dat
```

Agregar el final del archivo las siguientes líneas:

```
[compiz/profiles/unity/plugins/place]
position-x-values=[1366]
position-y-values=[0]
position-matches=['name=eog']
position-constrain-workarea=[true]
```

Guardar y cerrar. **NOTA:** el número 1366 es el tamaño horizontal de un monitor en particular, para saber qué número debe colocar en su caso utilice el comando

```
shell> xrandr -q
```

Y observe qué tamaño horizontal tiene el monitor primario.

■ fswebcam

Para instalar este programa abrir la ventana de comandos (*ctrl+t*) y escribir

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

El modo de operar está incluido en el módulo. La configuración de la captura se encuentra en el archivo `webcamConfig`².

²Para dudas sobre fswebcam consultar

<http://manpages.ubuntu.com/manpages/lucid/man1/fswebcam.1.html>

<https://github.com/fsphil/fswebcam/blob/master/example.conf>

3. Lista de funciones

A continuación se enlistan las funciones disponibles tras cargar el módulo. Nótese que algunas funciones pueden tener más de un método (aceptan más de un conjunto de argumentos). Nótese además que al cargar el módulo no solo se obtiene acceso a las funciones, también se define el valor de la constante `nombre_improbable_2pi` como el último número del archivo `Modulador/src/dospi`. 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.

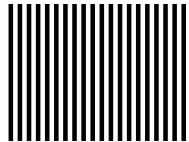

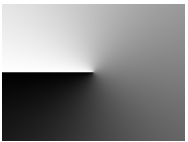
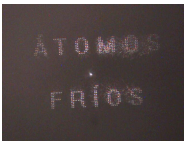
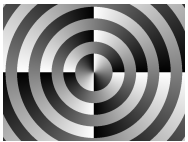
- `inicia()`
Esta función debe correrse al inicio de cada sesión si se va a trabajar con el SLM. Lo que hace es: configurar la salida del segundo monitor (conectada previamente al SLM) corriendo el código del archivo `PrepMonit1`, cargar la configuración de *eog* (ver sección 2) corriendo el código del archivo `PrepMonit2` y 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 en el segundo monitor ya que se abre con el programa *eog*.
- `finaliza()`
Esta función debe correrse al final de cada sesión en la que se haya utilizado la función `inicia`, ya que regresa la configuración de *eog* a su versión original.
- `faseMatInt(z::Matrix,gray2pi::Int64,gray0::Int64)`
Esta función toma como argumento una matriz cualquiera y la convierte en una matriz con entradas tipo `Int64` con valores que van desde `gray0` hasta `gray2pi`.
- `faseMatInt(z::Matrix,gray2pi::Int64)`
Si no se especifica el valor de `gray0` se toma igual a uno
(=`faseMatInt(z,gray2pi,1)`)
- `faseMatInt(z::Matrix)`
Si tampoco se especifica el valor de `gray2pi` se toma igual al valor de la constante `nombre_improbable_2pi`
(=`faseMatInt(z,nombre_improbable_2pi,1)`)
- `grayImage(matInt::Array{Int64,2})`
Esta función toma como argumento una matriz de enteros y regresa una imagen en escala de grises de 8-bits.
- `monitor2(imagen::Image)`
Esta función despliega en el segundo monitor (SLM) la imagen que toma por argumento.
- `escalon(nVer::Int64,nHor::Int64,fondo::Int64,dosPi::Int64, periodo::Int64)`
Esta función genera una rejilla binaria en forma de matriz de enteros de $nVer \times nHor$. La función tiene control sobre el periodo y profundidad de la rejilla con los últimos tres argumentos.
- `escalon(fondo::Int64,dosPi::Int64,periodo::Int64)`
Si no se especifican, las dimensiones serán de 800×600
(=`escalon(800,600,fondo,dosPi,periodo)`)
- `escalon(dosPi::Int64,periodo::Int64)`
Si no se especifica el fondo de la rejilla se tomará igual a uno
(=`escalon(800,600,1,dosPi,periodo)`)
- `escalon(periodo::Int64)`
Si tampoco se especifica el valor de `dosPi` se toma igual al valor de la constante `nombre_improbable_2pi`
(=`escalon(800,600,1,nombre_improbable_2pi, periodo)`)

- `blazeMat(nVer::Int64,nHor::Int64,dosPi::Int64,periodo::Int64)`
Esta función genera una rejilla de diente de sierra en forma de matriz de enteros de $nVer \times nHor$. La función tiene control sobre el periodo y profundidad de la rejilla con los últimos dos argumentos.
- `blazeMat(dosPi::Int64,periodo::Int64)`
Si no se especifican, las dimensiones serán de 800×600
(=`blazeMat(800,600,dosPi,periodo)`)
- `blazeMat(periodo::Int64)`
Si tampoco se especifica el valor de `dosPi` se toma igual al valor de la constante `nombre_improbable_2pi`
(=`blazeMat(800,600,nombre_improbable_2pi, periodo)`)
- `thetaMat(th)`
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. `th` corresponde a los grados por los cuales se puede rotar el holograma.
- `thetaMat()`
(=`thetaMat(0)`)
- `thetaMatInt(n,dosPi,th)`
Es casi igual a la función anterior pero regresa una matriz de enteros que van de 1 a `dosPi`.
- `thetaMatInt(n,dosPi)`
Si no se especifica un valor de rotación, el holograma se mantiene sin rotar
(=`thetaMatInt(n,dosPi,0)`)
- `thetaMatInt(n)`
Si tampoco se especifica el valor de `dosPi` se toma igual al valor de la constante `nombre_improbable_2pi`
(=`thetaMatInt(n,nombre_improbable_2pi,0)`)
- `funBesselJ(n,l,w,th)`
Esta función genera una matriz de 800×600 con entradas complejas que representan la amplitud compleja de la función de onda de un haz Bessel. Aquí `n,l` corresponden al orden de la función Bessel, `w` corresponde al ancho y `th` corresponde a los grados por los cuales se puede rotar el holograma.
- `funBesselJ(n,l,w)`
Si no se especifica un valor de rotación el holograma se mantiene sin rotar.
- `rapidBesselJ(n,l,w,th)`
Esta función genera una matriz de enteros que representan la fase de un haz Bessel
(=`faseMatInt(angle(funBesselJ(n,l,w,th)))`)
- `rapidBesselJ(n,l,w)`
Si no se especifica un valor de rotación, el holograma se mantiene sin rotar
(=`faseMatInt(angle(funBesselJ(n,l,w)))`)
- `capturaImg()`
Utilizando el programa *fswebcam* descrito en la sección 2, esta función realiza una captura a partir de una cámara usb y guarda la imagen en la carpeta `Modulador/capturas/`. La configuración de la captura se encuentra en el archivo `webcamConfig`.
- `calibrar()`
Esta función es para calibrar la relación *tono de gris – fase inducida* para una longitud de onda del láser. NOTA: Esta función no es exportada para evitar su uso accidental, ya que la calibración toma cerca de 20 minutos. Para utilizarla se tiene que llamar con preámbulo: `Modulador.calibrar()` .

Antes de mandar dichas imágenes es necesario utilizar la función `inicia()` que hace los ajustes necesarios para que la salida DVI (conectada al SLM) funcione como segundo monitor con resolución de 800×600 y despliega una imagen blanca 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(x::Image)`, la cual toma como argumento una imagen y la despliega en el segundo monitor. Al terminar de utilizar el programa se debe usar la función `finaliza()` que regresa los ajustes de monitor originales.

4. Ejemplos

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

In[]	<code>escalon(256,40)</code>	<code>blazeMat(256,80)</code>	<code>thetaMat(1)</code>	<code>capturaImg()</code>	<code>rapidBesselJ(2,2,16)</code>
Out[]					

Cuadro 1: Muestra de lo que hacen las funciones cargadas en el módulo. Las imágenes son resultado de aplicar la función `grayImage(x::Array{Int64,2})` (convierte una matriz de enteros a una imagen en escala de grises de 8-bits) a las funciones del renglón superior. Excepto en el caso de `capturaImg()` la cual corre un código de *Linux* basado en el programa *fswebcam* que permite capturar imágenes de una cámara con conexión USB y controlar sus propiedades (resolución, tiempo de exposición, escala de grises, etcétera).

Finalmente se muestra un ejemplo de los comandos utilizados para: abrir *Julia*, cargar el módulo, iniciar la comunicación con el controlador del SLM, mandar vórtices de distintos órdenes al SLM, finalizar comunicación y cerrar *Julia*. Nótese que para que esto funcione se debe haber realizado la configuración previa (sección 2) y debe estar conectado el controlador del SLM vía DVI.

```

shell> julia
...
julia> using Modulador
julia> inicia()
julia> monitor2(grayImage(thetaMatInt(0)))
julia> monitor2(grayImage(thetaMatInt(1)))
julia> monitor2(grayImage(thetaMatInt(2)))
julia> monitor2(grayImage(thetaMatInt(4)))
julia> finaliza()
julia> exit()
...
shell>

```