Obligatorio 2, Detección de bordes

28 de mayo de 2024

1. Generalidades

La aprobación de este curso se consigue mediante la correcta implementación de dos pequeños proyectos de programación (que llamaremos obligatorios). Éstos son propuestos en dos momentos del curso, aumentando en complejidad y forman parte de un mismo paquete, alimentándose mutuamente. Los programas desarrollados en la primer entrega serán utilizados en la segunda. Cada obligatorio será entregado a través de una página web habilitada para tales fines, con fecha límite de entrega señalada en la misma página. Estas entregas se complementan con un parcial escrito cuyo objetivo es evaluar aspectos más teóricos relacionados con el propio obligatorio.

Es importante recalcar que **tanto la prueba escrita como el proyecto entregado son individuales.** El sistema de recepción de entregas, además de almacenar los archivos enviados por los estudiantes, realiza un control de copias contra las entregas de otros estudiantes así como de programas similares que se encuentran en la web. Ese programa es capaz de detectar copias "maquilladas", es decir donde se cambiaron nombres de variables u otras formas de ocultar una copia. Este asunto debe ser bien entendido. No tenemos ningún inconveniente en que discutan soluciones, miren en la web, etc. pero **el trabajo entregado debe ser realmente el producto de vuestro trabajo y si el programa de control de copia detecta que hubo copia ello implica una sanción que puede significar la pérdida del curso e incluso sanciones mayores, tal como está especificado en el reglamento de la Facultad [1].**

El sistema intentará compilar y ejecutar la entrega de cada estudiante, a fin de dar un mínimo de información respecto de qué tan bien funciona. La evaluación preliminar mencionada anteriormente **no** determina la nota obtenida en la prueba, siendo ésta definida por una evaluación global por parte de los docentes que incluye los obligatorios, los parciales y la participación en clase.

El valor de cada obligatorio o parcial en el puntaje final del curso es el siguiente:

Prueba	valor en puntos sobre 100	puntaje mínimo
Obligatorio 1	30	8
Obligatorio 2	40	10
Parcial	30	8

Cuadro 1: Puntajes de cada prueba en PIE.

Además, la suma de todas las pruebas debe ser mayor o igual a 60 puntos.

1.1. Formato del archivo a entregar

El archivo entregado debe ser un archivo comprimido en formato **zip** (NO se aceptan archivos en formato rar), de nombre

nombres_separados_por_infraguiones.apellidos_separados_por_infraguiones.zip

y que los fuentes estén en la raíz del zip. El contenido del archivo debe incluir los siguientes elementos (que deben estar en la raíz del mismo y no en un directorio interno):

[1] `https://www.fing.edu.uy/es/gestion/normas-y-reglamentos`

- Todos los archivos fuente creados por el estudiante (**.h** y **.c**)
- Un archivo **Makefile** para compilar el o los programas requeridos en el trabajo.

Por ejemplo, supongamos que el obligatorio consiste en la generación de un ejecutable de nombre `obligatorio2`, su nombre es Juan Pablo Perez Fernandez, y usted implementó dicho ejecutable en un archivo **imagen.c** y su correspondiente archivo de encabezado **imagen.h**. Entonces debe subir un archivo de nombre:

`Juan_Pablo.Perez_Fernandez.zip`

con el siguiente contenido:

```
imagen.c
imagen.h
obligatorio2.c
Makefile
```

El **Makefile** en este caso debe ser así:

```
all: libimagen.a obligatorio2

DEPS = imagen.h  iio.h
LDLIBS = -ljpeg -ltiff -lpng

obligatorio1: obligatorio1.o libimagen.a
        cc $(CFLAGS) -o $@ obligatorio2.o -L./ -limagen -liio $(LDLIBS) -lm

%.o: %.c %.h $(LDLIBS)
        cc $(CFLAGS) -c $<

libimagen.a: imagen.o
        ar rcs $@ $<

clean:
        rm -rf *.o
        rm -rf libimagen.a
        rm -rf obligatorio2
```

En el caso de trabajar en Mac recientes (M1 o M2), el **Makefile** en este debe ser:

```
all: libimagen.a obligatorio1

DEPS = imagen.h iio.h

# Compiler flags
CFLAGS = -I<path-libs-estudiantes>/jpeg/include -I<path-libs-estudiantes>/libtiff/
    include -I<path-libs-estudiantes>/libpng/include
LDLIBS = -L<path-libs-estudiantes>/jpeg/lib -L<path-libs-estudiantes>/libtiff/lib
    -L/<path-libs-estudiantes>/libpng/lib -ljpeg -ltiff -lpng

obligatorio1: obligatorio1.o libimagen.a
        clang $(CFLAGS) -o $@ obligatorio2.o -L./ -limagen $(LDLIBS) -lm -liio

%.o: %.c $(DEPS)
        clang $(CFLAGS) -c $<

libimagen.a: imagen.o
        ar rcs $@ imagen.o

clean:
        rm -rf *.o
        rm -rf libimagen.a
        rm -rf obligatorio2
```

Nota: se sugiere transcribir manualmente el Makefile al archivo correspondiente, ***no*** *utilizar Copiar y Pegar para evitar que se introduzcan espacios u otros caracteres adicionales. Notar también que el Makefile es sensible a indentación, por lo que es necesario que se respete la tabulación de las líneas 5,7 y 10.*

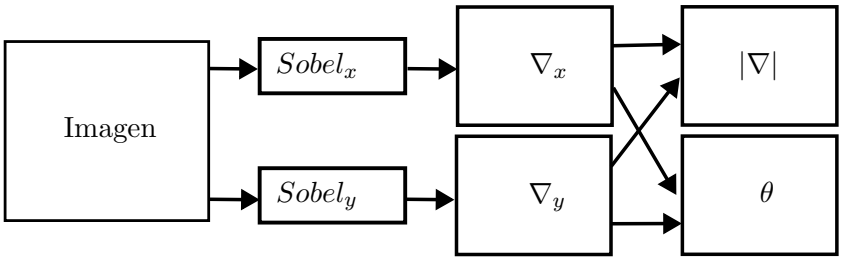

Figura 1: Diagrama en bloques del cálculo del gradiente de una imagen de entrada en escala de grises que produce las imágenes intermedias ∇_x,∇_y,$|\nabla|$ y θ.

Nota 2: si tienen una Mac reciente deben sustituir < path − libs − estudiantes > por el camino al lugar donde tienen las bibliotecas en su máquina.

En el archivo **Makefile** anterior se puede generar la biblioteca **libimagen.a** y también compilar un pequeño programa de prueba que hemos llamado **obligatorio2.c** que simplemente llama a las funciones implementadas con ciertos valores como parámetros de entrada e imprime el resultado. Esto les debe servir a ustedes para ver si dan los resultados correctos las funciones que están en la biblioteca. Para ello se debe invocar el **Makefile** de la siguiente manera:

```
make libimagen.a
make obligatorio2
```

La primera línea genera la biblioteca **libimagen.a** que se utiliza en el programa **obligatorio2.c**. La segunda línea genera el ejecutable **obligatorio2**.

En la ultimas líneas se incluye clean, que se encarga de borrar todos archivos generados durante la compilación. Lo pueden ejecutar de esta forma:

```
make clean
```

También pueden invocar todo (all):

```
make
```

Nota: Pueden crear un zip desde la máquina Linux/Mac con el comando `zip`; la sintaxis es, desde la carpeta de trabajo:

```
$zip nombre_archivo.zip imagen.c imagen.h obligatorio2.c Makefile
```

en el ejemplo anterior, sería `zip Juan_Pablo.Perez_Fernandez imagen.c imagen.h obligatorio2.c Makefile`

2. Introducción al problema

Este obligatorio continúa al anterior y se aplican los mismos conceptos relativos a la forma de representar las imágenes, así como al uso de la biblioteca de entrada salida `libiio.a`. Nos limitamos acá, entonces a explicar los conceptos relacionados con las tareas específicamente solicitadas en este obligatorio.

El objetivo de este obligatorio es implementar una serie de funciones que permitan detectar los bordes presentes en la imagen.

Para ello se procederá de la siguiente forma, representada gráficamente en la figura 1:

1. Filtrado de Sobel. El objetivo de esta etapa es calcular una estimación del gradiente ∇ de la imagen. Para ello, se aplica a la imagen monocromática dos filtros ($Sobel_x$ y $Sobel_y$), que producen las componentes horizontal y vertical del gradiente, respectivamente. El gradiente es un vector (∇_x, ∇_y) en cada punto de la imagen, que se calcula utilizando información local para diferenciar según el eje x y el eje y, respectivamente. El filtro es un núcleo $3x3$ de la forma:

$$Sobel_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

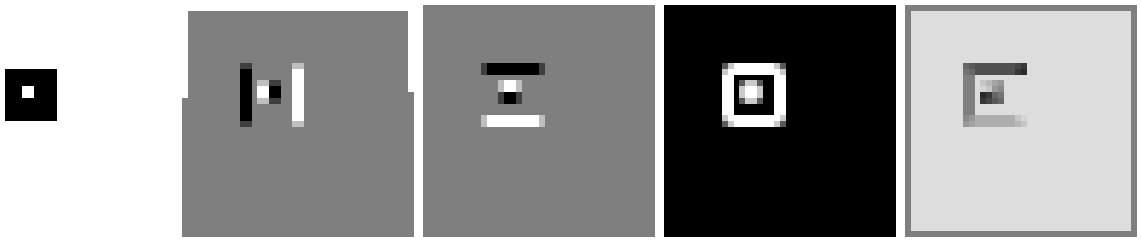

Figura 2: De izquierda a derecha: Una imagen de test en escala de grises formada por un cuadrado negro sobre fondo blanco y otro cuadrado, más pequeño, blanco adentro del anterior; la imagen de ∇_x, ∇_y, $|\nabla|$ y θ. Noten que para poder visualizar los resultados hubo que ajustar la salid de cada imagen de manera que el rango dinámico fuera entre 0 y 255 y así poder visualizarla. Por ejemplo, en el caso de ∇_x, ∇_y y θ, cuyos valores pueden ser negativos se agregó un valor de 127 de modo que ese valor equivale al 0, y los valores menores a 127 representan valores negativos. En el caso de θ, la fase del lado vertical derecho del cuadrado mayor es 0.

y

$$Sobel_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

que al aplicarse a la imagen produce la componente ∇_x del gradiente ∇ en ese punto. Por ejemplo, el gradiente según x en el punto p de coordenadas $(4, 7)$ se calcularía así:

$$\nabla_y(4,7) = I(5,6) + 2 * I(5,7) + I(5,8) - I(3,6) - 2 * I(3,7) - I(3,8)$$

donde $I(i, j)$ representa el valor de gris de la imagen en las coordenadas (i, j).

El gradiente puede representarse también como el vector $(|\nabla|, \theta)$ en cada punto. Donde $|\nabla|$ es el módulo del gradiente que se calcula:

$$|\nabla| = \sqrt{\nabla_x^2 + \nabla_y^2}$$

y la fase θ (en radianes) que indica la dirección del vector gradiente en dicho punto, que se calcula de la siguiente forma:

$$\theta = \arctan 2\nabla_y / -\nabla_x$$

Esta forma de representar el gradiente tiene la ventaja de ser más fácilmente interpretable. El módulo representa la magnitud del cambio de intensidad en ese punto y la fase indica la dirección del vector gradiente que corresponde con la dirección de mayor cambio local de intensidad (y por tanto es perpendicular a un borde en caso de pasar por ese punto).

Si queremos visualizar el gradiente tenemos el problema de que no es un escalar sino un vector con dos componentes ((∇_x, ∇_y) o $(|\nabla|, \theta)$), por tanto debemos generar dos imágenes, una para cada componente, que podemos visualizar. En caso de usar ese método recuerden tomar en cuenta el rango dinámico de la señal que están visualizando. Por ejemplo, la fase varía entre $-\pi/2$ y $\pi/2$ y si lo quieren visualizar en una imagen de unsigned chars deberán tomar eso en cuenta (los caracteres sin signo toman valores entre 0 y 255). Una opción es sumar un offset de 127 (que sería el cero) y además multiplicar la fase por algún factor que haga que el valor máximo sea menor a 127. En ese caso hay que recordar luego al visualizar la imagen, que los valores menores a 127 representan valores negativos de la fase, etc. Lo mismo se aplica para visualizar ∇_x y ∇_y, etc.

Nota1: Para visualizar la fase y que a todos nos de el mismo rango usaremos un factor 30, de esta manera:

$$Fase[ind] = 127 + (Fase[ind] * 30);$$

Nota: usaremos la misma estrategia de bordes que en el obligatorio 1, es decir, los píxeles que estén en la primera y últimas filas y columnas no serán tratados por el filtrado de Sobel.

2. Supresión de no máximos locales.

   Una vez que se ha obtenido el gradiente de la imagen (representada en cualquiera de sus formas (∇_x, ∇_y) o $(|\nabla|, \theta)$), procedemos a suprimir los no máximos locales. Los bordes son aquellos puntos en que $|\nabla|$ es máximo en la dirección θ, pero sucede que a veces en esa dirección hay más de un píxel con valor del $|\nabla|$ máximo. Es necesario entonces quedarnos solo con un máximo a lo largo de esa dirección. Véase la figura 3. Para ello se procede de la manera siguiente:

   - Se recorre la imagen de izquierda a derecha y de arriba a abajo.
   - En cada píxel en el cual $|\nabla| \neq 0$ (que llamaremos punto l de coordenadas (i, j)) se mira la fase θ y se calculan los puntos m y k cuyas coordenadas están determinadas por θ. Procedemos de la manera siguiente:
     - Si $\theta \geq 0$,
       - las coordenadas (ii, jj) del punto m serán

         $$ii = i - \sin(\theta) + 0{,}5$$

         y

         $$jj = j + \cos(\theta) + 0{,}5$$

       - y las coordenadas (ii, jj) del punto k serán

         $$ii = i + \sin(\theta) + 0{,}5$$

         y

         $$jj = j - \cos(\theta) + 0{,}5$$

     - Si $\theta < 0$
       - las coordenadas (ii, jj) del punto m serán

         $$ii = i - \sin(-\theta) + 0{,}5$$

         y

         $$jj = j - \cos(-\theta) + 0{,}5$$

       - y las coordenadas (ii, jj) del punto k serán

         $$ii = i + \sin(-\theta) + 0{,}5$$

         y

         $$jj = j + \cos(-\theta) + 0{,}5$$

     Recuerden que las funciones sin() y cos() dan valores entre 0 y 1. Para calcular las coordenadas debemos redondear, de allí el 0,5 que se adiciona, antes de convertir el resultado a enteros.

   Una vez calculadas las posiciones de los puntos vecinos según la dirección θ, m y k, comparamos el valor del módulo entre esos 3 puntos, y si no se cumple que $|\nabla(l)| > |\nabla(m)|$ y $|\nabla(l)| > |\nabla(k)|$ (el módulo en el punto l no es el mayor), entonces ese punto no es un máximo local y se apaga (es decir no es borde).

   Este procedimiento va a explorar todos los puntos potencialmente borde (o sea cuyo módulo del gradiente es grande) y al terminar el proceso habrá dejado en la imagen de bordes solo un máximo local del módulo en la dirección de θ.

3. Umbralización con histéresis. El valor de $|\nabla|$ a lo largo del borde varía. Ello depende del contraste local. Por otro lado, valores importantes del módulo de gradiente pueden ser generados por la presencia de un borde pero también pueden ser producidos por ruido. En la figura 4 puede observarse una imagen en niveles de gris y a su derecha una imagen binaria en la que están en negro todos los píxeles que son máximos locales de $|\nabla|$. Puede observarse que los bordes aparecen así como también muchísimo ruido. En términos generales podemos decir que cuando hay un $|\nabla|$ muy fuerte hay grandes chances de estar en presencia de un borde, pero si es un punto aislado con un gradiente importante, ello es quizás producto de ruido. Para resolver este problema, la forma de proceder en general es pasar un umbral sobre $|\nabla|$. Si el mismo es grande hay grandes chances de que ese punto sea parte de un borde.

   Observemos la segunda fila de la figura 4, la figura de la izquierda muestra el resultado de quedarnos solo con los puntos donde $|\nabla| > 15$, es decir puntos con un valor importante del módulo del gradiente. Se

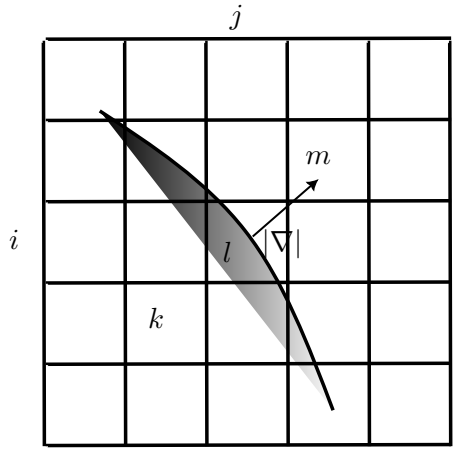

Figura 3: Ilustración del algoritmo de supresión de no máximos. El vector gradiente apunta en la dirección perpendicular al borde. El punto l está en las coordenadas (i, j). Según θ se fijan las coordenadas de los puntos m y k para comparar el valor de $|\nabla|$ en los puntos l, m y k.

observa que muchos de esos puntos pertenecen a los bordes de la imagen y que muchos puntos de ruido han desaparecido. Sin embargo existen puntos que pertenecen a los bordes y tienen un $|\nabla| < 15$ por lo que no aparecen en esta imagen. Por ejemplo los límites del rectángulo más oscuro que se ve en la parte interior del techo del auto. Si bajamos el umbral a 5, como se observa en la figura central, recuperamos esos puntos de borde pero a la vez dejamos pasar muchos puntos de ruido.

La solución de este problema es aplicar dos umbrales con histéresis. Definimos un umbral alto, que llamaremos $U^{sup}_{|\nabla|}$ y marcamos como puntos de borde todo píxel para el cual $|\nabla| > U^{sup}_{|\nabla|}$. En el ejemplo, si $U^{sup}_{|\nabla|} = 15$ esto corresponde a los puntos marcados en la figura de la izquierda. Luego marcaremos también como píxeles de borde, al mayor de los 8-vecinos de un píxel de borde y cumpla $|\nabla| > U^{inf}_{|\nabla|}$. Siguiendo con el ejemplo, si $U^{inf}_{|\nabla|} = 5$, esto permite marcar como píxeles de borde aquellos que aparecen en la segunda figura que están *conectados* con píxeles de alto contraste. El resultado de aplicar este método con $U^{inf}_{|\nabla|} = 5$ y $U^{sup}_{|\nabla|} = 15$ puede observarse en la figura de la derecha. Un píxel 8-vecino del píxel p es un píxel q que está a una distancia 1 en una grilla cuadrada. La figura 5 ilustra esto.

3. Descripción de la tarea

La implementación de esta tarea consiste en agregar a la biblioteca de tratamiento de imágenes creada en el obligatorio1 (`libimagen.a`), las funciones necesarias para realizar la detección de bordes.

La biblioteca consiste en dos archivos: un encabezado `imagen.h` y una implementación `imagen.c`, los cuales serán utilizados dentro del ejecutable a entregar. En `imagen.h` se deberán agregar las declaraciones de las nuevas funciones y en `imagen.c` se debe agregar la implementación de dichas funciones.

La biblioteca será utilizada como parte de la implementación del ejecutable llamado `obligatorio2`, pero también deberá poder ser invocada por un ejecutable no especificado.

4. Especificación de requerimientos

Se debe declarar las siguientes funciones en `imagen.h`, y definirlas en `imagen.c`:

- `CodigoError Sobel(const Imagen *pin, float *Gx, float *Gy, float *F, float *M, float *B)`.

  Esta función toma como entrada la imagen apuntada por **pin** y llena los siguientes arreglos de flotantes, de igual dimensión que la imagen, generados previamente fuera de la función:

  - G_x, que contiene los valores de la componente x del gradiente de la imagen en cada píxel, calculados por Sobel.
  - G_y, que contiene los valores de la componente y del gradiente de la imagen en cada píxel, calculados por Sobel.

Figura 4: De izquierda a derecha y arriba a abajo: Una imagen de test en escala de grises, el resultado del filtrado de Sobel y la eliminación de no mínimos locales (sin umbralizar), los bordes que quedan luego de aplicar un umbral $|\nabla| > 15$, los bordes que quedan luego de aplicar un umbral $|\nabla| > 5$ y los bordes que quedan luego de aplicar dos umbrales con histéresis $U^{inf}_{|\nabla|} = 5$ y $U^{sup}_{|\nabla|} = 15$.

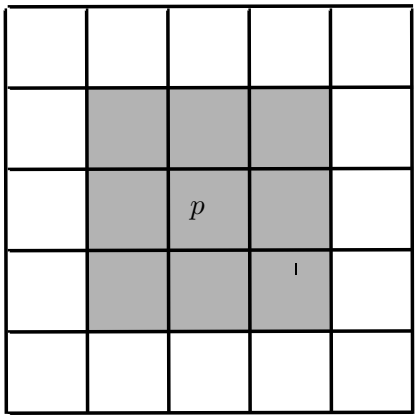

Figura 5: En gris los píxeles que son 8-vecinos del píxel p.

- **F**, que contiene los valores de θ en cada píxel, calculados por Sobel.
- **M**, que contiene los valores de $|\nabla|$ en cada píxel, calculados por Sobel.
- **B**, que contiene los valores de $|\nabla|$ en cada píxel, calculados por Sobel. En este arreglo irán los bordes. En esta etapa tiene los mismos valores que **M**, pero sus valores irán siendo modificados luego en las funciones `DosUmbralesConHisteresis` y `SupresionNoMaximos`.

Las dimensiones de la imagen, así como su arreglo de píxeles, están en **pin**. Devuelve un valor de tipo `CodigoError` según el caso.

Esta función aplica Sobel tal como se explicó más arriba y llena los valores del gradiente expresados como (G_x, G_y) y como $(|\nabla|, \theta)$.

- `CodigoError SupresionNoMaximos(float *M, float *F,float *B,int fil,int col)`. Esta función toma como entrada los siguientes arreglos de flotantes, de igual dimensión que la imagen:
  - **M**, que contiene los valores de $|\nabla|$ en cada píxel, calculados por Sobel.
  - **F**, que contiene los valores de θ en cada píxel, calculados por Sobel.
  - **B**, que contiene los valores de $|\nabla|$ en cada píxel, calculados por Sobel. En este arreglo iremos eliminando aquellos que no sean máximos locales y constituye la forma de devolver los píxeles candidatos a borde.

  Además tiene como parámetros dos enteros con el número de filas y el número de columnas de la imagen y también de los arreglos M, F y B. Devuelve un valor de tipo `CodigoError` según el caso.

  Esta función recorre todo el arreglo **B**, de izquierda a derecha y de arriba a abajo, y aplica las ideas explicadas anteriormente para eliminar de **B** aquellos puntos cuyo $|\nabla|$ no sea máximo local en la dirección de θ.

- `void DosUmbralesConHisteresis(float *M, float *B,int fil,int col, float` Th_b`, float` Th_a`)`.

  Esta función recibe los siguientes arreglos de flotantes, de igual dimensión que la imagen:
  - **M**, que contiene los valores de $|\nabla|$ en cada píxel, calculados por Sobel.
  - **B**, que contiene los valores de $|\nabla|$ en cada píxel, calculados por Sobel y posteriormente filtrados por la función `SupresionNoMaximos()`. En este arreglo estará la salida con los bordes finales.

  Recibe como parámetros también dos enteros con el número de columnas y el número de filas y dos flotantes Th_b y Th_a que corresponden a los umbrales $U^{inf}_{|\nabla|}$ y $U^{sup}_{|\nabla|}$, respectivamente.

  Esta función aplica el procedimiento explicado más arriba.

4.1. Programa del obligatorio

Deben escribir un programa, **obligatorio2.c**, que contenga solo el `main()` y que invoque las distintas funciones de la biblioteca. Prueben todas las funciones de la biblioteca verificando los resultados mediante la visualización de las imágenes que creen.

Interfaz de linea de comandos La sintaxis será por línea de comando de la siguiente manera:

```
./obligatorio2 archivoEntrada opcion valor
donde:
archivoSalida: contiene la ruta del archivo de entrada.
opcion:
G: convertir a gris
E: ecualizar
RG: ruido gaussiano [potencia]
RSP: ruido sal y pimienta [potencia]
H: histograma
A: histograma acumulado
P: filtro promediador [orden]
M: filtro mediana [orden]
B: deteccion de bordes
```

En el caso de que el parámetro opción sea RG o RSP, el tercer parámetro (valor) es un flotante que corresponde a la potencia.

En el caso de que el parámetro opción sea P o M, el tercer parámetro (valor) es un entero que corresponde al orden del filtro.

Los parámetros son:

- el primer parámetro es el nombre del archivo a leer, incluyendo el camino absoluto del directorio donde está el archivo (si fuera necesario). Archivo de entrada.
- el segundo parámetro es un carácter que debe indicar qué función probar según la siguiente nomenclatura:
  - G convertir a gris la imagen de entrada y poner el resultado en la de salida en una imagen llamada `ImagenGrises.png`.
  - E ecualizar la imagen de entrada (previa conversión a gris si es en color) y enviarla a la salida llamada `ImagenEcualizada.png`.
  - RG aplicar a la imagen de entrada (previa conversión a gris si es en color) un ruido Gaussiano cuya potencia está definida en el 3er parámetro. Guardar el resultado en un archivo llamado `ImagenConRuidoGaussiano.png`.
  - RSP aplicar a la imagen de entrada (previa conversión a gris si es en color) un ruido de sal y pimienta cuya potencia está definida en el 3er parámetro. Guardar el resultado en un archivo llamado `ImagenConRuidoSalYPimienta.png`.
  - H calcular el histograma de la imagen de entrada (previa conversión a gris si es en color) y guardarlo como un archivo de texto llamado `histo.txt`.
  - A calcular el histograma acumulado de la imagen de entrada (previa conversión a gris si es en color) y guardarlo como un archivo de texto llamado `histoAcum.txt`.
  - P aplicar el filtro promediador a la imagen de entrada (previa conversión a gris si es en color). El orden del filtro está definido por el 3er parámetro. Guardar el resultado en un archivo llamado `ImagenFiltradaPromediador.png`.
  - M aplicar el filtro de mediana a la imagen de entrada (previa conversión a gris si es en color). El orden del filtro está definido por el 3er parámetro. Guardar el resultado en un archivo llamado `ImagenFiltradaMediana.png`.
  - B: Aplicar la detección de bordes, según la secuencia:
    - `CodigoError Sobel(const Imagen* pin, float * Gx, float * Gy, float * F, float * M, float * B)`.
    - `CodigoError SupresionNoMaximos(float *M, float *F,float *B,int fil,int col)`.
    - `void DosUmbralesConHisteresis(float * M, float *B,int fil,int col, float Th_b, float Th_a)`.

    El resultado se guarda en sendos archivos llamados `Gradx.png`, `Grady.png`, `Modulo.png`, `Fase.png` y `Bordes.png`. **Nota2:** El Gradiente según x y según y, la Fase, el Módulo del Gradiente y los bordes son todos arreglos de flotantes. Para visualizar cualquiera de estos, utilizaremos la función `iio_write_image_float_vec(NombreArchivo, ArregloFloat, columnas, filas, 1)`, de la biblioteca `libiio.a` que toma el arreglo de flotantes *ArregloFloat*, de dimensiones *filas* por *columnas* y lo guarda en el archivo *NombreArchivo*.
- El tercer parámetro puede existir o no, dependiendo del valor del 2do parámetro. Su significado puede ser:
  - potencia valor flotante entre 0 y 1.
  - orden del filtro, entero entre 1 y 3.

Si el número de parámetros de la línea de comando es incorrecta debe imprimirse un mensaje explicando cómo realizar una invocación correcta y el significado de cada parámetro.

El archivo de entrada puede ser en colores. El programa debe actuar de manera acorde según si la entrada es a colores o blanco y negro. En el caso de que el archivo sea en colores, se debe convertir a grises y todas las funciones aplicarse sobre la versión en grises de la imagen. Si el archivo de entrada es en grises todas las funciones se aplican sobre la imagen tal cual (en grises).

Consideraciones y sugerencias

- Recuerden que si utilizan funciones matemáticas de **math.h** deben luego linkear con la biblioteca de matemática con la opción `-lm` al final de la linea que genera el ejecutable, para que dichas funciones estén definidas.
- Es **fundamental** liberar correctamente toda la memoria que haya sido reservada por el programa. **Parte de la evaluación del funcionamiento correcto de la tarea incluirá verificar que esto se esté haciendo correctamente**.
- Como siempre, implementar y probar de a poco.
- En caso de bugs, el GDB es su mejor amigo.