Filtrado de imágenes - Parte 1

28 de marzo de 2017

1. Generalidades

La aprobación de este curso se consigue mediante la correcta implementación de dos pequeños proyectos de programación. Éstos son propuestos aproximadamente un mes antes de cada parcial, y entregados a través de una página web habilitada para tales fines, con fecha límite de entrega fijada poco antes de cada período de parcial. Cada entrega es complementada con una pequeña prueba escrita 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.

En casos de ser posible, el sistema intentará además compilar y ejecutar la entrega de cada estudiante, de modo de dar un mínimo de información al respecto de qué tan bien funciona la entrega. Dependiendo del proyecto, esta evaluación preliminar estará o no disponible.

En todo caso, la evaluación preliminar mencionada anteriormente **no** determina la nota obtenida en la prueba, siendo ésta definida por una evaluación manual por parte de los docentes.

1.1. Formato del archivo a entregar

El archivo entregado debe ser un archivo comprimido en formato **zip** (NO se pueden subir rar), de nombre *apellido1_apellido2.nombre1_nombre2***.zip**. El contenido del archivo debe incluir los siguientes elementos:

- Todos los archivos fuente creados por el estudiante (**.h** y **.c**)
- La biblioteca precompilada entregada por los docentes, si es que existe.
- Un archivo **Makefile** para compilar el o los programas requeridos en el trabajo.

Por ejemplo, supongamos el obligatorio consiste en la generación de un ejecutable de nombre `oblig`, su nombre es Hernán Fabián Pérez González, y usted implementó dicho ejecutable en tres módulos **a.c** (y su correspondiente encabezado **a.h**), **b.c** (encabezado **b.h**) y **main.c**. Además, los docentes le entregaron una biblioteca auxiliar con un encabezado **aux.h** y código objeto **aux.o**. Entonces debe subir un archivo de nombre *perez_gonzalez.hernan_fabian.zip* con el siguiente contenido:

```
a.c
a.h
b.c
b.h
aux.h
aux.o
main.c
Makefile
```

El **Makefile** podría ser así:

```
oblig: main.o a.o b.o aux.o
  cc -o oblig main.o a.o b.o aux.o -lm

main.o: main.c
  cc -c main.c

a.o: a.c
  cc -c a.c

b.o: b.c
  cc -c b.c
```

Nota: Pueden crear un zip desde la máquina virtual con el comando `zip`; la sintaxis es

```
$zip -r nombre_archivo.zip carpeta_a_comprimir
```

en el ejemplo anterior, sería `zip -r` *perez_gonzalez.hernan_fabian.zip ./*
Suponiendo que en el directorio actual se encuentran los archivos a comprimir.

1.2. Metodología de trabajo

Algunas recomendaciones generales sobre cómo trabajar con proyectos como los que se proponen aquí:

- Simplicidad (KISS - Keep It Simple, Stupid). No complicar el código más allá de lo requerido.
- Prolijidad. No importa cuánto aburra, documentar bien lo que se hace es fundamental; es muy fácil olvidarse lo que uno mismo hizo. Esto incluye la inclusión de comentarios y el uso de variables con nombres autoexplicativos, si es posible.
- Incrementalidad. Implementar y probar de a pequeños pasos. "No construir un castillo de entrada". Es muy difícl encontrar las causas de un problema si se prueba todo simultáneamente.

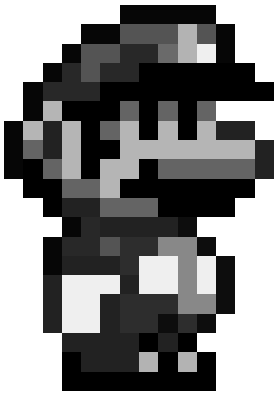
2. Introducción al problema

El problema que se plantea en este obligatorio, el procesamiento de imágenes, es una de las subáreas más populares e importantes del procesamiento de señales. Si bien desde el punto de vista teórico y formal las herramientas para trabajar con este tipo de problemas se ven recién en los últimos dos años de la carrera de Ingeniería Eléctrica, es posible trabajar con, y comprender informalmente, muchos algoritmos importantes de procesamiento de imágenes del estilo de los que se ven en aplicaciones populares de retoque de fotografías.

2.1. Representación digital de imágenes

Lo primero que tenemos que definir es cómo se representa una imagen digitalmente, a fin de poder almacenarla en memoria y trabajar con ella. Para simplificar el trabajo, nos limitaremos a imágenes en blanco y negro, técnicamente denominadas "de escala de grises", como la que se ve en la figura 1. La representación usual de este tipo de imágenes es la de una matriz de tamaño $m \times n$, donde m es el alto y n el ancho, en pixeles, de la imagen. Si denominamos como $\mathbf{X}$ a tal matriz, y medimos la posición de los pixeles de la imagen a partir de su esquina superior izquierda $(0, 0)$, el valor de la posición $\mathbf{X}(i, j)$ de la matriz correspondiente a esa imagen nos indicará la intensidad del pixel ubicado en la coordenada (i, j); mientras más alto el valor de $\mathbf{X}(i, j)$, más brillante será el pixel de la coordenada

Figura 1: Imagen en escala de grises.



255	255	255	255	255	255	6	6	6	6	6	255	255	255
255	255	255	255	6	6	84	84	84	179	84	6	255	255
255	255	255	6	84	84	40	40	106	179	239	6	255	255
255	255	6	40	84	40	40	0	0	0	0	0	0	255
255	6	40	40	40	0	0	0	0	0	0	0	0	0
255	6	181	0	0	0	100	0	100	0	100	255	255	255
6	181	34	181	0	100	181	0	181	0	181	34	34	255
6	100	34	181	0	0	181	181	181	181	181	181	181	34
6	0	100	181	0	181	181	0	100	100	100	100	100	34
255	0	0	100	100	181	0	0	0	0	0	0	0	255
255	255	0	34	34	100	100	100	0	0	0	0	255	255
255	255	255	6	40	34	34	34	34	13	255	255	255	255
255	255	6	40	40	84	45	45	136	136	13	255	255	255
255	255	6	34	34	34	45	239	239	136	239	13	255	255
255	255	34	239	239	239	34	239	239	136	239	13	255	255
255	255	34	239	239	34	45	45	45	136	136	13	255	255
255	255	34	239	239	34	45	45	13	45	13	255	255	255
255	255	255	34	34	34	34	0	34	0	255	255	255	255
255	255	255	0	34	34	34	179	0	179	0	255	255	255
255	255	255	0	0	0	0	0	0	0	0	255	255	255

Figura 2: Izq.:Imagen en escala de grises de 20×20 pixeles. Der.: pixeles de la imagen

(i, j) en la imagen. En este proyecto nos limitaremos a imágenes de 8 bits por pixel, que en general se traduce a que $\mathbf{X}(i, j)$ puede tomar valores entre 0 y 255, siendo 0 el color *negro* y 255 el *blanco*. La figura 2 da un ejemplo de lo anterior.

Hasta el párrafo anterior vimos cómo representar una imagen como una matriz de números. Resta entonces definir cómo representar esa matriz en memoria. Hay básicamente dos maneras: mediante arreglos multidimensionales, o mediante arreglos unidimiensionales de algún tipo numérico (por ejemplo `int` o `float`). La primera es posiblemente la más natural, pero depende de arreglos multidimensionales en C, los cuales a esta altura no han sido vistos en el curso y por ende no manejaremos en este trabajo.

En el caso unidimiensional, es necesario definir cómo se traduce cada coordenada bidimensional (i, j) de la imagen a un índice unidimensional dentro del arreglo y viceversa. El método que utilizaremos en este trabajo, llamado "barrido por filas", almacena la fila superior en los primeros n elementos del arreglo (recuérdese que en C los índices comienzan en 0, por lo cual el pixel $(0, 0)$ irá en la posicón 0 del arreglo y el pixel $(n - 1, 1)$ en la $n - 1$). La segunda fila ocupa los siguientes n elementos (de n a $2n - 1$), y así por delante hasta completar los $m \times n$ elementos del arreglo.

2.2. Filtrado de imágenes

Habiendo definido cómo se representan las imágenes en memoria, vayamos a lo que es el objetivo de este proyecto, es decir, el filtrado de imágenes. Por *filtro* entendemos cualquier función que tome una imagen como entrada y devuelva otra como salida, de manera que esta última es una versión

alterada de la primera. Existe una enorme cantidad de filtros y efectos de procesamiento de imágenes; algunos tienen como objetivo mejorar las imágenes (como por ejemplo eliminar ruido o aumentar el contraste y la nitidez), distorsionar o aplicar efectos artísticos, resaltar partes (por ejemplo bordes y esquinas) o transformar (rotar, escalar). El objetivo de este proyecto es el de implementar algunos de estos filtros y aplicarlos a un conjunto de imágenes que serán dadas.

En esta primera etapa trabajaremos con imágenes de tamaño fijo de 512×340 pixeles. Dichas imágenes serán dadas como archivos PGM, un formato de archivos muy sencillo. Para leer y guardar dichas imágenes se utilizará una pequeña biblioteca realizada por los docentes, de modo que la tarea a realizar por los estudiantes se concentre en la implementación de los filtros.

3. Descripción de la tarea

Concretamente, la tarea consistirá en aplicar los filtros descritos a continuación a un conjunto de imágenes predefinido. Todos los filtros serán implementados e invocados desde un único programa ejecutable de nombre `obligatorio`, el cual deberá generarse como resultado de la compilación del proyecto entregado a través de la utilidad `make` (para lo cual debe disponerse de un archivo `Makefile` apropiado). La sintaxis de línea de comandos para la ejecución de cada filtro debe ser la siguiente:

```
./obligatorio comando parametro entrada.pgm salida.pgm
```

donde `comando` es el nombre del filtro a aplicar, `parametro` es el parámetro numérico del algoritmo (si lo hubiera, de otro modo puede ser cualquier cosa, **pero debe estar siempre presente**), `entrada.pgm` la imagen de entrada al programa y `salida.pgm` el nombre de la imagen de salida del programa. Tanto para la lectura como para la escritura de archivos PGM, se contará para esta entrega con una pequeña biblioteca de funciones implementada por los docentes. Esta biblioteca consta de dos archivos: **pgm.h**, con la declaración de las funciones de la biblioteca (a utilizar en el `#include`), y **pgm.o**, con la implementación de dichas funciones, ya compilada para **Linux 32 bits** (para aquellos que utilicen Linux de 64 bits, podemos proveer una versión bajo solicitud).

El uso de dichas funciones está documentado en el propio archivo `pgm.h`, como es usual en la práctica de programación moderna.

Para la realización de la tarea se entregará una serie de archivos, descritos a continuación. Estos archivos deberían estar en el mismo directorio en donde se ubiquen los archivos creados por el alumno.

- `pgm.h` y `pgm.o` encabezado e implementación de rutinas de lectura/escritura de archivos PGM. Esta versión está (artificialmente) restringida a leer archivos PGM de 8 bits de 512×340 pixeles.
- `imagenes.zip` conjunto de imágenes PGM de 512×340 pixeles de prueba; debe descomprimirse de modo que las imagenes queden en el mismo directorio que el ejecutable.
- `probar.sh` este es un *script* (secuencia de comandos de UNIX) que compilará el proyecto entregado (mediante la invocación de la utilidad `make`) y luego ejecutará el programa generado sobre el conjunto de imágenes otorgado. Para que este script funcione, deben cumplirse los requisitos anteriormente descritos (todo debe estar en la misma carpeta), y debe existir además un archivo **Makefile** creado por el alumno para compilar el proyecto mediante la utilidad `make`.

3.1. Copia

Este filtro simplemente copia la imagen de entrada en la de salida, pixel a pixel, $\mathbf{Y} = \mathbf{X}$.

3.2. Negativo

Una categoría de filtrado de imágenes consiste en el *mapeo de colores*. Aquí la imagen de salida tiene el mismo tamaño que la de entrada, y el color de cada pixel de la imagen de salida se calcula como el resultado de una única función escalar del color del pixel de las mismas coordenadas en la

imagen de entrada. Uno de los más sencillos es el *negativo*, que transforma colores claros en oscuros y viceversa. Si trabajamos con imágenes de 8 bits, la entrada es $\mathbf{X}$ y la salida $\mathbf{Y}$, tendremos que $\mathbf{Y}(i,j) = 255 - \mathbf{X}(i,j)$ para toda coordenada (i,j) de la imagen.

3.3. Reflejo

Este filtro lo que hace es reflejar el contenido de la imagen. Vamos a considerar tres variantes del reflejado.

- **reflejo horizontal:** $\mathbf{Y}(i,j) = \mathbf{X}(i, n-j-1),\ 0 \le i < m,\ 0 \le j < n.$
- **reflejo vertical:** $\mathbf{Y}(i,j) = \mathbf{X}(m-i-1, j),\ 0 \le i < m,\ 0 \le j < n.$
- **reflejo central:** $\mathbf{Y}(i,j) = \mathbf{X}(m-i-1, n-j-1),\ 0 \le i < m,\ 0 \le j < n.$

3.4. Estiramiento de contraste

Muchas veces las imágenes son tomadas en malas condiciones, lo que produce que tengan poco contraste y por ende que sean poco visibles. Una de las formas más básicas de (intentar de) mejorar una imagen es la de tratar de "estirar" su rango de colores de modo que se utilice toda la gama posible de tonos disponible. En imágenes de 8 bits, esto se traduce en hacer que las partes más oscuras de la imagen sean negro puro (valor 0) y la más claras sean blanco puro (valor 255).

El procedimiento consiste en dos etapas bien sencillas. En el primero se determinan los valores máximos y mínimos de luminosidad de la imagen:

$$\begin{aligned} x_{\text{máx}} &= \max_{i,j} \mathbf{X}(i,j) \\ x_{\text{mín}} &= \min_{i,j} \mathbf{X}(i,j) \end{aligned}$$

En la segunda etapa, se define el valor de cada pixel en la imagen de salida de modo que el rango de colores original se mapee linealmente al rango $[0, 255]$:

$$\mathbf{Y}(i,j) = \frac{255}{x_{\text{máx}} - x_{\text{mín}}} \left(\mathbf{X}(i,j) - x_{\text{mín}}\right).$$

En caso de que $x_{\text{máx}} = x_{\text{mín}}$ (cosa poco probable), se copia la salida a la entrada.

3.5. Promedio de ventanas

Un método muy común de filtrado de imágenes lo constituye el *filtrado por ventanas*. Sea $\mathbf{Y}$ la imagen de salida de un filtro; supongamos que es del mismo tamaño que la imagen de entrada $\mathbf{X}$. El filtrado por ventanas calcula $\mathbf{Y}(i,j)$ como función de una *ventana* de pixeles de $\mathbf{X}$ en un entorno de las coordenadas (i,j). Un caso típico es el de ventanas cuadradas, donde el entorno de $\mathbf{X}(i,j)$ está formado por los pixeles de coordenadas (i',j') de modo que $i - w \le i' \le i + w$ y $j - w \le j' \le j + w$ para un valor w determinado; esto arroja una ventana de tamaño $N = (2w+1)\times(2w+1)$. En el caso del promedio de ventanas, tenemos

$$\mathbf{Y}(i,j) = \frac{1}{N} \sum_{i'=i-w}^{i+w} \sum_{j'=j-w}^{j+w} \mathbf{X}(i',j').$$

Más en general puede decirse que el píxel de salida es una suma ponderada de los píxeles del vecinaje, donde la ponderación define las características del filtro. En este caso tenemos:

$$\mathbf{Y}(i,j) = \sum_{i'=i-w}^{i+w} \sum_{j'=j-w}^{j+w} \mathbf{W}(i'-i, j'-j)\mathbf{X}(i',j').$$

Nótese que en esta expresión desaparece la división por N, para que esto sea correcto la normalización debe incluirse dentro de la *máscara*. Nótese también que hay una doble suma y que esta se aplica sobre una zona de la imagen, centrada en el punto (i,j) y un vecinaje de longitud w. Por su lado la *máscara* es cuadrada de dimensión $(2w+1) \times (2w+1)$ y se recorre a la vez. Para el caso del filtro promediador mencionado antes, todos los valores de w (también llamada *máscara*, *núcleo* o *kernel*) valen 1:

$$\mathbf{W} = \begin{matrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{matrix}$$

Con esa *máscara* el resultado del filtro es la suma de los píxeles de la ventana dividido por el número de píxeles en dicho entorno. Esta formulación permite definir otros filtros lineales (donde el resultado es la suma ponderada de los valores del vecinaje), simplemente modificando la *máscara* w. Por ejemplo, si queremos hacer un filtrado que de mayor importancia al píxel central podemos definirla así:

$$\mathbf{W} = \begin{matrix} 1/9 & 2/9 & 1/9 \\ 2/9 & 4/9 & 2/9 \\ 1/9 & 2/9 & 1/9 \end{matrix}$$

Si queremos aplicar un filtro que desenfoque la imagen (operación conocida por su nombre en inglés *blur*), podemos aplicar un filtro lineal con una *máscara* cuyos valores surgen de calcular una gaussiana de parámetro σ centrada en el centro de la *máscara*. Para ello podemos calcular el valor de la gaussiana en cada punto usando una fórmula del tipo $A \times e^{\frac{-dx}{2\times\sigma^2}}$, donde A es una normalización y dx es la distancia al centro de la gaussiana, en este caso al centro de la *máscara*. Para el caso de $\sigma = 1{,}0$ obtenemos la siguiente *máscara*:

$$\mathbf{W} = \begin{matrix} 0{,}822025 & 1{,}01118 & 0{,}822025 \\ 1{,}01118 & 1{,}66716 & 1{,}01118 \\ 0{,}822025 & 1{,}01118 & 0{,}822025 \end{matrix}$$

Una observación interesante es que es posible definir una función *filtrado lineal* que tome como parámetros una imagen y una *máscara* y simplemente cambiando los valores del segundo parámetro (la *máscara*), se esté cambiando el filtro aplicado a la imagen.

Estrategia de bordes: Si no hacemos nada al respecto, tendremos un problema al calcular $\mathbf{Y}(i,j)$ para valores de i o j que estén más cerca que w de los bordes de la imagen, ya que en ese caso habrán valores de i' y j' que caerán fuera de la imagen $\mathbf{X}$. Para estos casos se define una *estrategia de borde*. En el obligatorio utilizaremos la técnica de "extender el borde", que consiste en tomar el pixel de borde más cercano a la posición fuera de la imagen para la cual queremos inventar el valor. Esto se reduce a hacer un *clipping* de las coordenadas (ver Práctico 2, ejercicio "Frecuencias, histogramas y probabilidades empíricas"):

$$i'' = \text{clip}(i', 0, m)$$
$$j'' = \text{clip}(j', 0, n)$$

y luego utilizar el valor de $\mathbf{X}(i'', j'')$ en lugar de $\mathbf{X}(i', j')$.

3.5.1. Ejercicios complementarios

Escribir un programa general en el que, utilizando una sola función *filtrado lineal*, se puedan aplicar los siguientes filtros:

1. Filtro promediador
2. Filtro gaussiano. Se trata de un filtro pasabajos donde los valores de la *máscara* son aproximaciones de una gaussiana.

3.6. Detector de bordes

La detección de bordes es una etapa fundamental en la detección de objetos en aplicaciones relacionadas con la denominada "visión por computadora". Además de eso, suele usarse con fines estéticos. En este trabajo veremos una forma muy sencilla de hacer esto, que se reduce esencialmente a calcular las derivadas direccionales de la imagen en sentido vertical y/o horizontal. Sea $\nabla_x \mathbf{X}$ la derivada horizontal discreta de $\mathbf{X}$; esta es una imagen del mismo tamaño que $\mathbf{X}$ donde el pixel (i, j) contiene el valor de la derivada de $\mathbf{X}$ en ese mismo punto. Sea $\nabla_y \mathbf{X}$ la derivada vertical. Hay muchas formas de calcular estas imágenes; nosotros utilizaremos la siguiente:

$$\begin{aligned} \nabla_x \mathbf{X}(i,j) &= \mathbf{X}(i,j+1) - \mathbf{X}(i,j-1) \\ \nabla_y \mathbf{X}(i,j) &= \mathbf{X}(i+1,j) - \mathbf{X}(i-1,j) \end{aligned}$$

(**La *estrategia de bordes* a seguir es la misma que en el filtro de promedio de ventanas anteriormente descrito.**) Con estas dos imágenes podemos definir el gradiente de la imagen $\mathbf{X}$ mediante el mapeo

$$\nabla \mathbf{X}(i,j) \to (\nabla_x \mathbf{X}(i,j), \nabla_y \mathbf{X}(i,j)).$$

Más aún, podemos calcular ahora el módulo del gradiente $|\nabla \mathbf{X}|$ como la imagen

$$|\nabla \mathbf{X}|(i,j) = \sqrt{\nabla_x \mathbf{X}(i,j)^2 + \nabla_y \mathbf{X}(i,j)^2}.$$

Notar que tanto $\nabla_y \mathbf{X}$ como $\nabla_x \mathbf{X}$ y $\nabla \mathbf{X}$ pueden contener valores fuera del rango $[0, 255]$. Es importante pues que si se desea representar cualquiera de ellas tres como imagen, el resultado sea mapeado a ese rango previamente. En el caso de $|\nabla \mathbf{X}|$, que siempre es no negativo, sencillamente se trunca cualquier valor por encima de 255 al valor 255.

En caso de querer representar $\nabla_y \mathbf{X}$ y $\nabla_x \mathbf{X}$, que pueden tomar valores negativos, una posiblidad es la de tomar su valor absoluto, por ejemplo $|\nabla_y \mathbf{X}|$, y luego truncarlo para que no supere 255.

Para obtener las derivadas horizontales y verticales de la imagen ($\nabla_y \mathbf{X}$ y $\nabla_x \mathbf{X}$) podemos aplicar filtros lineales con una *máscara* w adecuada. Por ejemplo podemos utilizar el llamado *filtro de Sobel*, definido por las siguientes •emphmáscaras:

$$\mathbf{W_{SOBEL_H}} = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

y

$$\mathbf{W_{SOBEL_V}} = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

Notar que en este caso el filtro aplica a la vez una derivada en un sentido y un filtrado pasabajo en el sentido perpendicular. Para observar los resultados es conveniente realizar un estiramiento de los datos de salida del filtro, a fin de garantizar que el rango de los valores de la imagen procesada (por ejemplo el gradiente en un sentido), cubra la totalidad del rango de niveles de gris disponibles (o a 255).

3.7. Especificación de requerimientos

1. Implementar el filtro de copia (comando `copia`) en una función de nombre `copia` que tome la siguiente lista de parámetros, en el sigiuente orden:
   - arreglo pixels de entrada,
   - ancho de la imagen
   - alto de la imagen
   - arreglo de pixels de salida
2. Implementar el filtro de negativo (comando `negativo`) en una función de nombre `negativo` que tome la siguiente lista de parámetros, en el siguiente orden:
   - arreglo pixels de entrada,
   - ancho de la imagen
   - alto de la imagen
   - máximo valor de pixel
   - arreglo de pixels de salida

   Al finalizar la función, el arreglo de pixels de salida contendrá la imagen negativa de la ingresada en el arreglo de pixels de entrada.
3. Implementar el filtro de reflejo (comando `reflejo`) en una función de nombre `reflejo` que tome la siguiente lista de parámetros, en el siguiente orden:
   - arreglo pixels de entrada,
   - ancho de la imagen
   - alto de la imagen
   - parámetro (puede ser 0, 1 o 2).
   - arreglo de pixels de salida

   Al finalizar la función, el arreglo de pixels de salida contendrá el reflejo de la ingresada en el arreglo de pixels de entrada. Si el parámetro vale 0, la imagen será reflejada horizontalmente. Si vale 1, se reflejará verticalmente. Si vale 2, se reflejará en ambos sentidos (siendo así una simetría central). En otro caso se imprimirá un mensaje de error y se volverá sin realizar ningún cálculo.
4. Implementar el estirado de contraste (comando `contraste`), en una función de nombre `contraste` que tome la siguiente lista de parámetros, en el siguiente orden:
   - arreglo pixels de entrada,
   - ancho de la imagen
   - alto de la imagen
   - máximo valor de pixel
   - arreglo de pixels de salida
5. Implementar la detección de bordes (comando `bordes`), en una función de nombre `bordes` que tome la siguiente lista de parámetros, en el siguiente orden:
   - arreglo pixels de entrada,
   - ancho de la imagen
   - alto de la imagen

- máximo valor de pixel
- parámetro (puede ser 0, 1 o 2).
- arreglo de pixels de salida

Si el parámetro vale 0, el arreglo de pixels de salida contendrá a $|\nabla_x \mathbf{X}|$ (**valor absoluto** truncado a $[0, 255]$). Si el parámetro vale 1, el arreglo de pixels de salida contendrá a $|\nabla_y \mathbf{X}|$ (**valor absoluto** truncado a $[0, 255]$). Si el parámetro vale 2, el arreglo de pixels de salida contendrá a $|\nabla \mathbf{X}|$ (**módulo del gradiente** truncado a $[0, 255]$). En otro caso se imprimirá un mensaje de error y se volverá sin realizar ningún cálculo.

6. Implementar el promedio de ventanas (comando `promedio`), en una función de nombre `promedio` que tome la siguiente lista de parámetros, en el siguiente orden:

   - arreglo pixels de entrada,
   - ancho de la imagen
   - alto de la imagen
   - máximo valor de pixel
   - parámetro valor entero mayor a 0
   - arreglo de pixels de salida

   El parámetro en este caso especifica el radio w de la ventana rectangular a aplicar. Si $w = 0$, el arreglo de pixels de salida debe contener una copia exacta del arreglo de los pixels de entrada al finalizar la función.

7. Implementar la función `main` que invoque a cada uno de los filtros anteriores, con los parámetros correspondientes, de acuerdo a los argumentos de entrada en la línea de comandos, tal como se explicara en la sección anterior. Más concretamente, la función main debe:

   - Leer los argumentos de línea de comandos y definir de allí los valores del tipo de filtro a aplicar, parámetro, archivo de entrada y archivo de salida.
   - Cargar la imagen de entrada utilizando las funciones provistas en **pgm.h**
   - Invocar al filtro especificado sobre la imagen de entrada.
   - Guardar la imagen de salida utilizando las funciones provistas en **pgm.h**

Consideraciones y sugerencias

- Se sugiere definir los arreglos para los pixeles de entrada y salida como variables globales de tipo arreglo de `int`.
- Ninguna de las funciones de filtrado devuelve un valor, por lo tanto deben declararse con valor de retorno `void`.
- Recuerde que puede utilizar la función `strcmp` de **string.h** para comparar cadenas de texto.
- 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.