Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Tesis de licenciatura

Implementación y fundamentación de los algoritmos de detección de bordes en imágenes digitales basados en la derivada

Presentado por:
C. Marco Antonio Espinoza Ramírez

Para obtener el grado de:
Licenciado en Matemáticas Aplicadas

Director de Tesis:
M. C. Sergio Adán Juárez

Puebla, Pue. Méx. 4 de octubre de 2023

Resumen

La presente tesis se enfoca en la Implementación y fundamentación de los algoritmos de detección de bordes en imágenes digitales basados en la derivada. En un mundo cada vez más orientado hacia la tecnología digital y la visión por computadora, la detección de bordes en imágenes desempeña un papel crucial en una amplia gama de aplicaciones, desde el procesamiento de imágenes médicas hasta la visión artificial. Este estudio aborda la implementación detallada de algoritmos de detección de bordes basados en la derivada, así como su fundamento teórico, explorando las ventajas y limitaciones de estos métodos en la extracción precisa de contornos en imágenes digitales. A través de una revisión exhaustiva de la literatura, se examina la evolución de estos algoritmos, se comparan sus resultados y se evalúa su desempeño en diversos contextos, proporcionando así una contribución valiosa al campo de procesamiento de imágenes digitales y visión por computadora.

Dedicatoria

"*A aquellos que aman el conocimiento.*"

Agradecimientos

Agradezco a mis padres: Martín e Irma por guiarme, apoyarme y amarme incondicionalmente desde el comienzo de mi vida. A mis abuelos, a mis tías y tíos, en especial a mi tía Paty por su cariño, su apoyo y sus consejos.

A mis profesores que con esmero comparten sus conocimientos y sus experiencias. A mis amigos: Uriel, Jesús, Miguel y Ulises por su apoyo y su compañía.

Al M. C. Sergio Adán Juárez por guiarme pacientemente en todo este proceso del desarrollo de esta tesis, al Dr. José Rubén Conde Sánchez por su aporte a este documento. Y a la BUAP por proporcionarme acceso a una educación integral y de calidad.

Introducción

La detección de bordes es una técnica fundamental en el procesamiento de imágenes y visión por computadora. Su objetivo es identificar los límites o transiciones significativas de intensidad en una imagen, que generalmente corresponden a bordes o contornos de objetos. Sirve de base para aplicaciones como la segmentación y reconocimiento de objetos, aplicaciones médicas, compresion de imágenes, seguridad y vigilancia, etc.

En esta tesis se exponen los fundamentos básicos de la teoría matemática que hace posible la implementación de los algoritmos de detección de bordes. El material bibliográfico que se puede encontrar al respecto no suele preocuparse por dar una explicación detallada del por qué funcionan. Tal vez la naturaleza aplicada de estos temas empuje a autores y estudiantes a marcar distancias con las formas rigurosas de las matemáticas. Por otro lado, quienes disfrutan del rigor de las matemáticas más puras, a menudo hacen de lado las posibilidades de su aplicación.

0.1. Motivación

La presente tesis surge con la motivación de robustecer el vínculo entre los fundamentos matemáticos y su aplicación práctica en el campo del procesamiento digital de imágenes. En este contexto, nuestro interés primordial radica en ofrecer una exposición accesible y enriquecedora de los conceptos matemáticos que yacen en la base de la implementación de algoritmos de detección de bordes basados en la derivada. A medida que exploramos la intrincada complejidad de estos métodos, nuestro propósito fundamental es proporcionar claridad y comprensión, contribuyendo así a un enriquecimiento tanto del conocimiento teórico como de su aplicación en esta fascinante

intersección entre la teoría matemática y la tecnología de imágenes digitales.

0.2. Justificación

La presente tesis se fundamenta en la creciente relevancia del procesamiento digital de imágenes en numerosos ámbitos, desde la medicina hasta la industria del entretenimiento, donde la detección de bordes desempeña un papel crucial en la identificación de objetos y la segmentación de imágenes. En este contexto, la motivación subyacente a este trabajo radica en la necesidad imperante de comprender y aplicar de manera efectiva los conceptos matemáticos que respaldan los algoritmos de detección de bordes basados en la derivada. Estos algoritmos, aunque fundamentados en teoría matemática sólida, a menudo pueden parecer inaccesibles o abrumadores para quienes no están familiarizados con los aspectos matemáticos subyacentes. Por lo tanto, la tesis busca establecer un puente entre la teoría y la práctica, ofreciendo una exposición clara y accesible de los principios matemáticos esenciales, con el fin de empoderar a los profesionales y estudiantes interesados en el procesamiento de imágenes digitales. Al hacerlo, se pretende enriquecer tanto la comprensión teórica de estos métodos como su aplicabilidad en situaciones del mundo real, fomentando así un avance significativo en la eficiencia y precisión de las técnicas de detección de bordes en imágenes digitales. En última instancia, esta investigación contribuye a fortalecer el conocimiento y la capacidad de innovación en el emocionante campo de la visión por computadora y el procesamiento de imágenes digitales, ofreciendo herramientas prácticas y teóricas para abordar los desafíos y oportunidades que se presentan en este ámbito en constante evolución.

0.3. Objetivos

0.3.1. Objetivo general

Investigar, analizar y proporcionar una comprensión profunda de los algoritmos de detección de bordes en imágenes digitales basados en la derivada, a través de la implementación detallada y la fundamentación teórica de estos métodos.

0.3.2. Objetivos particulares

1. Investigar y analizar los principios matemáticos y teóricos que subyacen a los algoritmos de detección de bordes basados en la derivada, con el fin de comprender a fondo su fundamento teórico.

2. Realizar una revisión detallada de la literatura existente en el campo de la detección de bordes en imágenes digitales, identificando y evaluando los métodos más relevantes y sus aplicaciones en diversas áreas.

3. Implementar de manera práctica y detallada una variedad de algoritmos de detección de bordes basados en la derivada en un entorno de procesamiento de imágenes, utilizando herramientas y lenguajes de programación adecuados.

0.4. Organización de la tesis

La presente tesis esta organizada por cinco capítulos:
El primer capítulo explica el concepto de imágenes digitales, la estructura interna y su representacion matemática para poder manejarlas de manera correcta.
El segundo capítulo aborda las nociones elementales del concepto de derivada con el rigor que requiere, proporcionando fundamentos que se espera sean lo suficientemente claros para más adelante iniciar su implementación computacional.
El capítulo tres hace uso de los elementos contenidos en los dos capítulos previos para construir los algoritmos de detección de bordes.
En el capitulo cuatro corresponde a la metodologia, aparece el análisis referente a la implementación de los algoritmos y las dificultades sorteadas.
Finalmente, el capítulo cinco contiene los resultados de aplicar estos algoritmos a una colección de imágenes.
Se implementaron los algoritmos en el lenguaje de programación C. En el apéndice se muestra un extracto de las funciones codificadas con su respectiva descripción. Se espera que para el lector sea un material útil y le sirva como punto de partida para realizar sus propias pruebas y conclusiones.

Índice general

Índice de figuras

Capítulo 1

Sobre las imágenes digitales

'Hay dos grandes grupos de imágenes digitales: las imágenes vectoriales y las imágenes de mapa de bits'(Oltra y Mellado, 2008). Si bien en la misma obra ambos autores hacen hincapié en la clara distinción que hay entre las imágenes que han sido tomadas de una fuente "tangible"hacia un medio intangible como lo es un entorno digital, llamándolas **imágenes digitalizadas** y refiriéndolas como una forma inferior frente a las llamadas **imágenes digitales**; una familia de imágenes que han nacido totalmente en el entorno digital y nunca han sido tangibles, también hacen a un lado esta distinción para tratar de destacar una característica más objetiva y menos filosófica como lo es, el mecanismo lógico por el cual un sistema digital almacenará y reconstruirá dichas imágenes.

Quien esté acostumbrado a tratar con imágenes digitales en mayor o menor medida, encontrará familiar el concepto de **imagen vectorial** y sus ventajas, son fácilmente escalables y suelen ser más ligeras, sin embargo las imágenes que obtenemos con métodos de escaneo o fotografía, antes llamadas digitalizadas, son casi siempre **mapas de bits**. Y como es de esperarse, la visión por computadora tiene preferencia por este tipo de imágenes provenientes de la realidad tangible.

1.1. Los mapas de bits

Las imágenes de mapas de bits, también llamadas usualmente bitmaps, pueden imaginarse como rejillas cuadriculadas, donde cada cuadrícula co-

Figura 1.1: Ejemplo de una imagen digitalizada y de una imagen digital.

rresponde a un pixel que es a la vez la unidad mínima de color homogéneo que constituye una imagen digital según (Oltra y Mellado, 2008), es decir, cada cuadro de la cuadricula está caracterizado por un color en específico, en conjunto y si son suficientes, los pixeles pueden reconstruir una imagen almacenada en memoria de forma digital en algo apreciable por el ojo humano, a través de una pantalla o mediante papel impreso.

Se puede comenzar a explorar el concepto de bitmap si se consideran las características importantes de su estructura que determinan el tipo y la calidad de la imagen que contienen, estas son: **la resolución**, **el color** y **la compresión**.

1.1.1. La resolución en la imagen digital

La resolución es la primera idea que puede venir a nuestra mente cuando nos preguntamos por la calidad de una imagen, elementalmente se basa en la cantidad de pixeles que hay en ellas. Suponga que una imagen tiene 800 pixeles de ancho y 600 de alto, entonces la imagen posee $800 * 600 = 480,000$ pixeles o en un termino más común 0,48 megapixeles.

Sin embargo, hay una característica más que hay que tener en cuenta cuando se quiere entender el concepto de resolución y es el pixel por pulgada **PPP** (o PPI; pixels per inch). Esto, como su nombre indica, es la cantidad de pixeles que hay por pulgada en una imagen, de modo que si nuestra imagen anterior de $800 * 600$ tuviera también un PPI de 200, la imagen tendría unas medidas de 4 pulgadas de ancho por 3 de alto.

Naturalmente, esto tiene que ver con los procesos de reconstrucción de la imagen (pasar de los datos almacenados en memoria a una imagen entendible por el ojo humano), con lo que, si la imagen se imprime con los PPI antes mencionados tendrá ese tamaño en papel, así mismo si se proyecta en un monitor tendrá esas medidas dentro del mismo. Los PPI son entonces una propiedad del medio de visualización, si por ejemplo visualizamos una imagen en un monitor con un PPI de 50 nuestra imagen medirá $16 * 12$ pulgadas, bastante más grande pero también se observará con baja calidad.

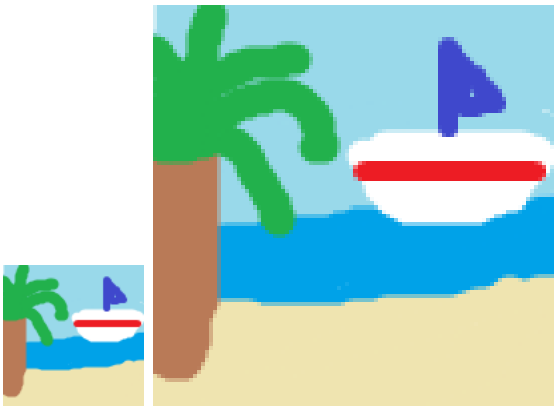

Figura 1.2: Comparación de la misma imagen a diferentes escalados.

1.1.2. El color en la imagen digital

La profundidad de color se refiere a la forma en la que un mapa de bits codifica el color, es evidente que una cantidad determinada de bits únicamente abarca una cantidad finita de colores. Si se destina sólo un bit por pixel, este representa dos colores: blanco o negro. En (Oltra y Mellado, 2008) presentan la siguiente lista:

- 1 bit de color ($2^1 = 2$ colores) monocromático: blanco y negro.
- 2 bits de color ($2^2 = 4$ colores).
- 4 bits de color ($2^4 = 16$ colores) profundidad mínima de color aceptada por el estándar VGA.
- 8 bits de color ($2^8 = 256$ colores) también llamado SuperVGA.

- 16 bits de color ($2^{16} = 65536$ colores) color de alta resolución o HiColor.
- 24 bits de color ($2^{24} = 16,777,216$ colores) color verdadero o TrueColor.
- 32 bits de profundidad corresponde a imágenes de 24 bits con 8 bits adicionales usados para información no cromática que permiten simulación de transparencia.

Además de la cantidad de memoria utilizada para codificar los colores en una imagen, también hay que tener en cuenta los diferentes mecanismos que suelen usarse para este fin, de entre las opciones que existen están el **tramado de difusión**, el **monocromático**, la **escala de grises**, los modelos **RGB** y **CMYK**, el **Duotono o Tritono** y el **HSI** (Sucar y Gómez, 2011) (Oltra y Mellado, 2008).

De manera breve se menciona que una imagen **monocromática** es la forma de color más simple que puede haber, los pixeles tienen una profundidad de 1 bit por lo que solo pueden representar blanco o negro, dando lugar a imágenes muy contrastadas, una posible solución a este problema es el llamado tramado de difusión, que aprovecha los PPI de una imagen para simular diferentes niveles de gris variando las densidades de pixeles de color negro sobre un fondo blanco.

Figura 1.3: De izquierda a derecha; imagen digital a color, imagen monocromática e imagen en escala de grises.

La **escala de grises** en cambio es más conocida y es usada por ejemplo como medio de impresión. Se compone de 8 bits por pixel, permitiendo 256 niveles de gris diferentes, siendo 0 el color negro y 255 blanco, veáse la imagen (X). Por otro lado los **duotonos y tritonos** permiten imprimir imágenes en escalas de grises con dos o tres tintas diferentes del gris.

El modelo **CMYK** (cyan, magenta, yellow, black) es el modo de color predominante en el medio impreso, está basado en el sistema de los colores primarios, que le dan su nombre, se usa generalmente a partir de mezclar las tintas de esos colores para obtener nuevos y por eso esta siempre presente en las imágenes impresas, elementalmente es un sistema sustractivo pues al añadir tintas se va incrementando la cantidad de luz que esta absorbe lo que acercará la mezcla al negro cada vez más. Adicionalmente aunque el color negro puede obtenerse a partir de los primeros tres, se suele tener a la mano una tinta negra ya que el negro generado por los demas colores no es el adecuado (Oltra y Mellado, 2008).

El modelo **RGB** (red, green, blue) funciona a partir de mezclar los tres colores que le dan nombre para generar nuevos colores, pero en un sentido aditivo, a diferencia del CMYK que resta información a la onda de luz, el RGB trabaja sumando información a la onda a partir de diferentes niveles de sus colores base, por lo que añadir más color a una mezcla hará que esta se aproxime al blanco.

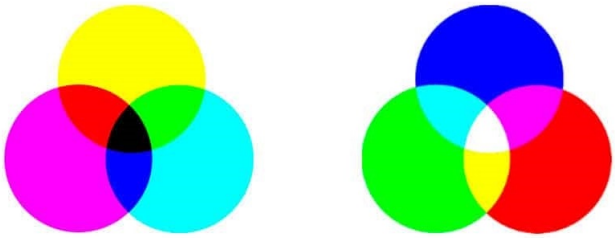

Figura 1.4: Ala izquierda el modo de color CMYK, a la derecha el modo RGB.

De acuerdo con (Sucar y Gómez, 2011) el **HSI** es el que más se aproxima a la visión humana, se caracteriza por tres componentes, Intensidad (I), Croma (H, hue) y Saturación (S). Si consideramos las diferentes combinaciones de color RGB dentro de un cubo donde cada componente de un punto en el espacio es un nivel de color para rojo, verde y azul. El HSI podría entenderse como un cilindro, donde la altura es la intensidad, la distancia al eje central la saturación y el ángulo el croma.

1.1.3. Formatos de compresión

Existen diversas maneras de almacenar los datos de un bitmap en la memoria de un ordenador, aunque hoy en día la memoria de la que dispone uno, es considerablemente grande, en los inicios de las imágenes digitales esta era terriblemente escasa, por lo que se desarrollaron diferentes mecanismos para almacenar las imágenes con la menor memoria posible.

Dentro de los métodos de compresión existentes, podemos encontrar compresión con y sin perdida, esto significa que almacenar una imagen mediante cierto algoritmo de compresión podría resultar en una perdida de calidad en mayor o menor medida. Algunos de los formatos de compresión para bitmaps más populares son;

JPG ó JPEG (Joint Photographic Expert Group) es un formato de compresión que utiliza un complejo algoritmo con perdida de información (Cruz Contreras, González Robles, y Herrera Lozada, 2004), lo que lo convierte en uno de los que más comprime, admite una profundidad de color a partir de 8 bits y hasta 24. Es el formato más indicado para imágenes a todo color con muchos matices, usado para imágenes que requieran calidad fotográfica.

PCX (PC Paint brush) es un formato que comprime el archivo. Consiste en reemplazar las secuencias de N pixeles consecutivos con el mismo color por dos bytes, el primero indica el número N y el segundo el color. No funciona muy bien en imágenes con mucho ruido, llegando a incrementar el tamaño del archivo.

GIF (Graphic Interchange Format) solo admite profundidades de color de 1 a 8 bits, utiliza un algoritmo de compresión convencional que detecta las repeticiones de ciertas secuencias de datos, tuvo mucha aceptación en los inicios de Internet pero fue desplazado por el PNG debido a que este ultimo tiene un algoritmo de dominio publico. El GIF también es capaz de manejar transparencias, aunque toscas, y contener animaciones básicas. Luego de un periodo de tiempo en el olvido, ha habido un renacimiento del formato al ser usado en las animaciones de WhatsApp.

PNG (Portable Network Graphics) apareció directamente para ser una

version mejorada del GIF, permiten profundidades de hasta 24 bits, permite el manejo de imágenes con muy alto contenido cromático ademas de transparencias. Este formato es el preferido para la web, iconos y emoticónos.

TIFF (Tagged Image File Format) es un formato sin perdida, ideal para imágenes que están destinadas a ser impresas pues los navegadores web no suelen reconocer este formato.

BPM (Windows BitMap) es el formato más simple que hay, Tiene la gran ventaja de ser sencillo, por lo que puede ser considerado como universal a pesar de necesitar mucho espacio en memoria. Este formato es el que utilizaremos durante nuestra implementación por lo que le dedicaremos una sección completa (Oltra y Mellado, 2008) (Cruz Contreras y cols., 2004).

1.1.4. El formato BMP

Un archivo **BMP** consiste fundamentalmente en tres partes, **Encabezado**, **Paleta de colores** y **Mapa de bits**. Como lo usual es que todas las imágenes manejen una profundidad de color de 24 bits y para este caso no se requiere una paleta de colores ya que esta se asume dada por la combinación de RGB en valores de 0 a 255, daremos más importancia a describir la estructura de la cabecera y el mapa de bits (Cruz Contreras y cols., 2004).

La cabecera es una serie de 54 bytes al inicio del archivo que proporciona información sobre el mismo para facilitar su lectura, la estructura es;

- 2 bytes con los caracteres "BM"
- 4 bytes que indican el tamaño total del archivo
- 4 bytes reservados
- 4 bytes que indican el offset desde el comienzo el archivo hasta el inicio del mapa de bits
- 4 bytes reservados
- 4 bytes que indican el ancho de la imagen (en pixeles)
- 4 bytes que indican el la altura de la imagen (en pixeles)

- 2 bytes que indican la cantidad de planos en la imagen
- 2 bytes que indican la cantidad de bits por pixel
- 4 bytes que indican la compresión
- 4 bytes que indican el tamaño de la imagen
- 4 bytes que indican la resolución horizontal
- 4 bytes que indican la resolución vertical
- 4 bytes que indican el tamaño de la tabla de color
- 4 bytes que indican el número de colores importantes

Después de la cabecera, para imágenes de 24 bits de profundidad, se encontrarán inmediatamente los datos del mapa de bits. Estos se encuentran organizados del byte menos significativo al más significativo.

Adicionalmente, la imagen esta invertida, de modo que al inicio se encuentra la linea final de pixeles y al final la primera, acomodadas de izquierda a derecha con una longitud múltiplo de 4, en caso de que la longitud difiera se agregan los bytes faltantes (Cruz Contreras y cols., 2004).

Cada pixel se encuentra compuesto de 3 bytes siendo el primero el nivel de azul (B), seguido por el nivel de verde (G) y finalmente el nivel de rojo (R), representados por números de 0 a 255 haciendo uso del modelo RGB. Es por eso que podemos considerar una imagen BMP como tres mapas de bits superpuestos, cada uno de los colores base (RGB) puede ser entendido como una imagen en escala de grises con tintas respectivas que resultan en una imagen colorizada al superponerlas.

1.2. Representación matemática de una imagen

Con el fin de desarrollar herramientas matemáticas para el procesar las imágenes es necesario construir un modelo matemático que represente el concepto que tenemos de imagen. (Sucar y Gómez, 2011) sugieren definir una

Figura 1.5: De izquierda a derecha, plano de color Rojo, plano de color Azul y plano de color Verde.

imagen como una función $f : \ A \subset R^2 \rightarrow B \subset R^3$ por lo que vamos a ir en esta dirección.

Como paso intermedio, para establecer claramente el significado de las expresiones que vamos a usar, es necesario dar la definición de función, aunque las definiciones pueden ser diversas y dado que algunos textos especializados pueden llegar a proveer definiciones bastante abstractas, se opta por presentar la definición que se da en (Angoa, Contreras, Ibarra, Linares, y Martínez, 2008).

Definición 1 *Una* ***función*** *es una regla que asocia a cada elemento de un cierto conjunto A un único elemento de un conjunto B. Al conjunto A suele llamarsele* ***dominio de la función*** *y al conjunto B* ***codominio*** *(o* ***contradominio****)* ***de la función****.*

Para indicar que f es una función con dominio A y codominio B se escribe:

$$f \ : \ A \rightarrow B$$

Lo usual es asociar un sistema coordenado (x, y) a la imagen con el origen en el extremo superior izquierdo, téngase en cuenta que esto es previo a la creación de un mapa de bits, por lo que el sistema coordenado sera un sistema continuo, que podemos caracterizar por $I \subset R^2$.

Luego para cada punto (x, y) de la imagen vamos a asociar un nivel de brillo para cada color, que para poder adecuar a nuestro sistema BMP será un

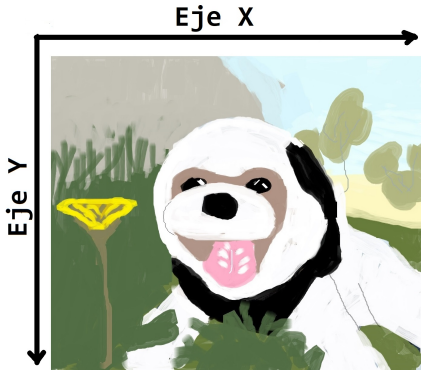

Figura 1.6: Imagen sobre un sistema cartesiano.

número en el rango de $[0, 255]$, de modo que vamos a disponer de 3 funciones;

$$f_R : I \rightarrow [0, 255]$$

$$f_G : I \rightarrow [0, 255]$$

$$f_B : I \rightarrow [0, 255]$$

Quien tenga algo de experiencia en cálculo vectorial sabrá que podemos resumir estas en una sola función $f : I \rightarrow [0, 255]X[0, 255]X[0, 255]$ que parte de un punto del plano de la imagen y devuelve un vector con tres valores reales (r, g, b) siendo cada uno el nivel de brillo de cada una de estas componentes de color en cada punto del plano imagen.

Como ultimo punto, téngase en cuenta que una imagen en escala de grises o en monocromático al manejar un solo plano reduce su complejidad al de una función de dos variables en una, esto es $f : I \rightarrow [0, 255]$. De modo que cada punto del plano de la imagen esta asociado por la función a un nivel de gris.

1.2.1. Representación matemática de una imagen digital

(Sucar y Gómez, 2011) describen una imagen digital como una imagen que ha sido discretizada tanto en valor de intensidad como espacialmente mediante un mapeo. Cada valor de intensidad es mapeado a un número entero entre 0 y 255, también, es necesario que el espacio sea mapeado a una

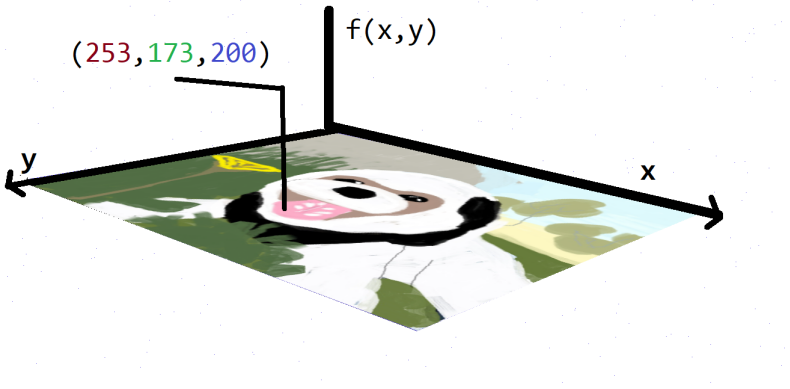

Figura 1.7: Vista tridimensional de una imagen sobre un sistema cartesiano, cada punto del sistema tiene asociado un vector de tres dimensiones.

cuadricula, que, puede ser representada por una matriz $N * M$, donde cada valor es un número que representa el nivel de intensidad correspondiente en la imagen.

Figura 1.8: Resultado de modificar la resolución de una imagen digital.

En la figura 1.8 la primera imagen (izquierda) tiene una resolución de $20 * 16$ mientras que la de la segunda (derecha) tiene una de $100 * 84$, lo que muestra que una cantidad mayor de pixeles al momento de realizar un muestreo incrementa la fidelidad de una imagen con respecto al medio real o analógico. Sin embargo es importante tener presente que al realizar la digitalización se ha perdido información en menor o mayor medida.

Con esto hecho, nuestra representación matemática para la imagen discretizada es naturalmente una función discretizada que asocia cada casilla de

una matriz $N*M$ con un entero entre 0 y 255 para imágenes monocromáticas o con un vector (r, g, b) igualmente compuesto por enteros para imágenes a color.

$$f : I \cap N^2 \to [0, 255] \cap N$$
$$f : I \cap N^2 \to [0, 255]^3 \cap N^3$$

1.3. Convolución

La convolución es una de las aplicaciones más interesantes e importantes de el modelo matemático anterior, es elementalmente un algoritmo que, a partir de ajustar ciertos parámetros, permite procesar imágenes mediante la aplicación de una variedad ilimitada de filtros. Una definición formal de esta la encontramos en (Morales, s.f.).

Definición 2 *Dada una matriz A_{nxm} y una matriz $C_{(2N+1)X(2N+1)}$ con $2N+1 < m, n$ se define la **convolución** de las matrices A y C como una nueva matriz $D = A * C$ definida a partir de la expresión*

$$d_{ij} = \frac{1}{c} \sum_{r=1}^{2N+1} \sum_{s=1}^{2N+1} a_{i-N+r-1, j-N+s-1} c_{r,s}$$

donde $c = \sum_{i,j=1}^{2N+1} c_{i,j}$ (si $c = 0$ se toma $c = 1$). Observe que $d_{i,j}$ sólo está definido para $i = N+1, ..., m-N-1$ y $j = N+1, ..., n-N-1$.

La matriz C se le denomina núcleo o kernel de la convolución. Algunos autores pueden referirse a la matriz C como mascara de convolución.

Aunque la definición anterior puede parecer intimidante, el proceso es relativamente simple, si se considera por ejemplo una matriz de convolución pequeña, digamos $3x3$ y una matriz que represente una imagen a procesar también pequeña ($5x5$) la convolución consiste en:

- Superponer la matriz de convolución a la imagen
- Multiplicar los valores que coincidan al quedar superpuestas
- Sumar los resultados de cada multiplicación

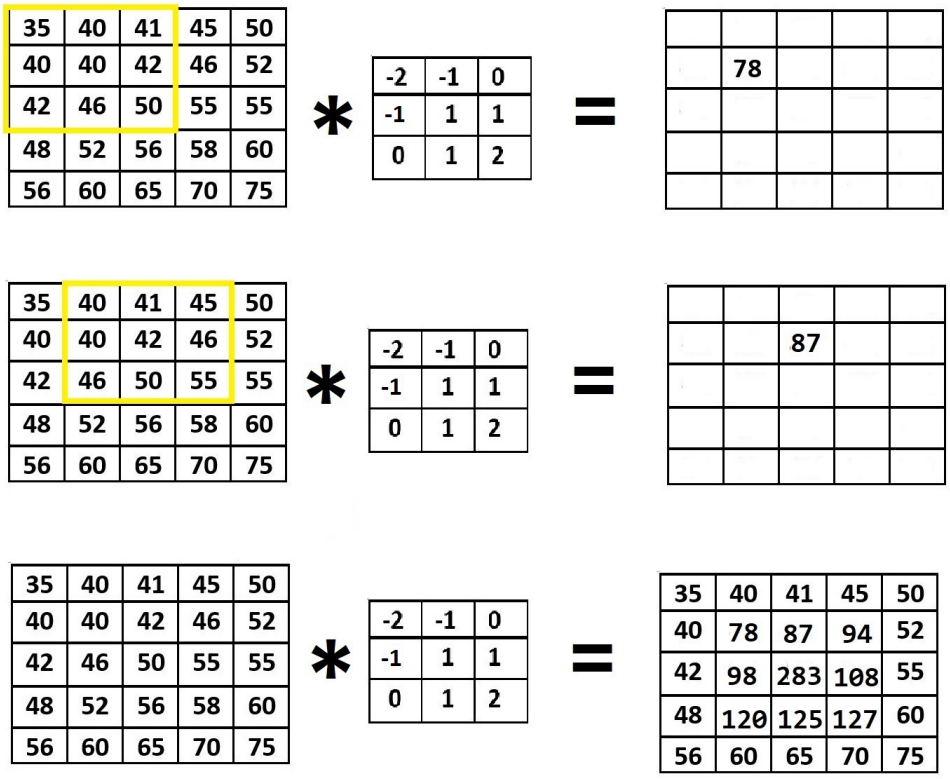

Figura 1.9: Ejemplo gráfico de la convolución en pequeña escala.
(Morales, s.f.)

- Dividir entre el factor c definido antes, (dependerá de la matriz de convolución)
- El resultado de esta operación se guarda en otra matriz, en la misma posición que el centro de la matriz de convolución.
- Mover la matriz de convolución una casilla en la dirección que se desee barrer la imagen

Si se presta atención, notará que los bordes de la matriz imagen no pueden ser procesados pues estos no tienen vecinos que encajen con la mascara de

convolución, por lo que la matriz resultante sera más pequeña. Una practica común es dejar estos bordes idénticos a la imagen fuente vea la figura 1.10 (Morales, s.f.).

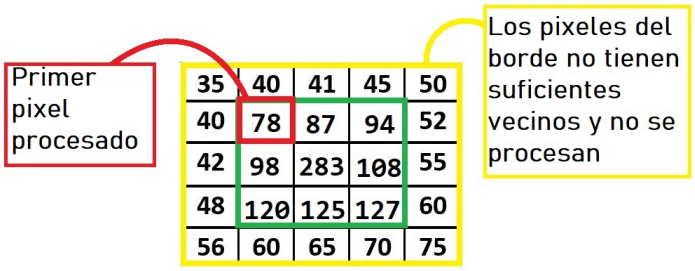

Figura 1.10: Representación de una imagen cuyos bordes no se han procesado.

Capítulo 2

Diferenciación

La mayoría de los algoritmos que existen para la detección de bordes en imágenes digitales, tal vez todos, están basados en el concepto del gradiente, que no es más que una generalización en varias variables del concepto de diferenciación en una sola. Por lo que se dedicara este capítulo a explorar, explicar y definir este concepto fundamental sobre el cual se desarrolla la teoría de detección de bordes.

2.1. Definición de Límite

Hablar de diferenciación en una variable puede tratarse de un tema trivial o no dependiendo del individuo, el objetivo de este apartado es establecer una base clara del concepto de derivada para el desarrollo del tema de interés en los capítulos posteriores, por lo que se ha decidido comenzar desde la definición de límite e intentar desarrollar el concepto brevemente , para ello se cita textualmente la definición proporcionada en (Spivak, 1988).

Definición 3 *Dada f una función, se dice que f* ***tiende hacia*** *l* ***en*** *a si y solo si para todo $\epsilon > 0$ existe algún $\delta > 0$ tal que, para todo x, si $0 < |x - a| < \delta$, entonces $|f(x) - l| < \epsilon$.*

En (Spivak, 1988) se muestra que una función no puede tender a dos números distintos como límite, lo que sustenta el uso de la notación $\lim_{x \to a} f(x) = l$ para referirse a ese único número l, la expresión se leería **El límite de** f **cuando** x **tiende a** a **es igual a** l.

En cuanto al concepto de límite, puede resumirse bajo la idea de que el valor en el codominio de la función asociado a determinado valor en el dominio se aproxima a un número l, siempre que el valor en el dominio se aproxime a a, un número que no necesariamente esta en el dominio.

Para simplificar el concepto y abusando del hecho de que se hace uso de este en un contexto que involucra R y R^2, y más específicamente en este capítulo a R. Se puede tomar como apoyo la figura 2.1, extraída del mismo libro que la definición.

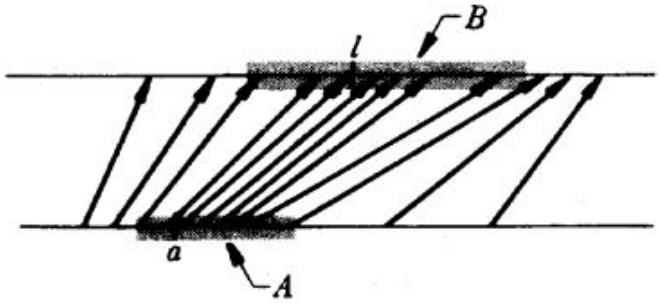

Figura 2.1: Ejemplo gráfico de la definición de límite presentado por Spivak. (Spivak, 1988)

El la figura de arriba se puede observar c'omo la función est'a representada por flechas que van del segmento inferior **(dominio)** al segmento superior **(co-dominio)**. También es posible apreciar a a y l y un par de regiones sombreadas al rededor de estos señaladas por A y B respectivamente, estas sombras simbolizan una región de puntos cercanos a a (y a l). Hay que prestar especial atención al hecho de que las flechas que parten del segmento de puntos cercanos a a apuntan a valores ubicados en el área sombreada alrededor de l.

2.2. Definición de derivada

Habiendo establecido la definición de límite, estamos listos para dar directamente la definición de derivada de manera inmediata y como aparece en (Spivak, 1988).

Definición 4 *Una función f es **derivable en** a si*

$$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

existe.
*En este caso, el límite se designa por $f'(a)$ y recibe el nombre de **derivada de f en a***
*Se dice también que f es **derivable** si lo es para todo su dominio.*

Así mismo deben hacerse un par de comentarios respecto a esta ultima definición para que tenga sentido con la notación que se usa en los ámbitos más prácticos del cálculo.

El primero es hacer notar que la derivada esta definida en un punto del dominio de la función y esta es un límite, es decir, un único valor. Sin embargo suele entenderse a la derivada de una función como una función por si misma, esto no es más que una pequeña confusión entre el termino $f'(a)$ y f', la primera denota el valor del límite antes descrito para la función f en el punto a, la segunda una nueva función que asocia cada punto del dominio donde f es derivable con su respectiva derivada en cada punto.

El segundo comentario es respecto al concepto geométrico de **tangente** a la gráfica de f en el punto $(a, f(a))$. Esto es, la recta que pasa por $(a, f(a))$ y tiene pendiente $f'(a)$, concepto que es de utilidad para comprender mejor el concepto de derivada. Es necesario pasar por este paso para entender adecuadamente el principio mediante el cual vamos a determinar los bordes en una imagen.

2.2.1. Interpretación de la derivada

Si bien existen diversas maneras de entender la derivada bajo contextos más cercanos a lo cotidiano (por ejemplo implicando el concepto de velocidad instantánea) para nuestros intereses una interpretación geométrica sobre la gráfica de una función sera lo mejor. Considérese la gráfica de una función, como la que modela el nivel de gris de una imagen analógica sobre una linea recta. Lo usual es que la gráfica tenga una forma similar a la de la figura 2.2.

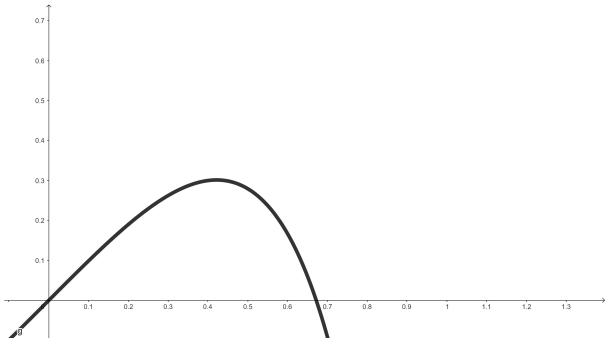

Figura 2.2: Ejemplo de la gráfica de una función.

Estamos interesados en la pendiente de la recta tangente a la la gráfica de f en el punto $(x, f(x))$. Es claro que un movimiento sobre el eje X corresponderá a un movimiento sobre el eje Y, que acaba convirtiéndose en un recorrido sobre la gráfica, digamos pues que el movimiento sobre el eje X recorre la gráfica de f. Podemos aproximar el valor de dicha pendiente, con la expresión

$$m = \frac{f(x+h) - f(x)}{h}$$

. Esto es, calcular la pendiente de una falsa tangente que corta por dos puntos separados por una distancia h, esto son, x y $x + h$.

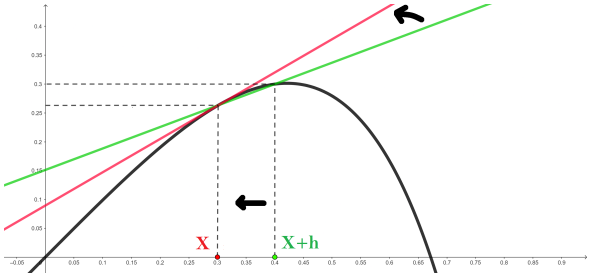

Figura 2.3: Aproximación a la recta tangente a una función en un punto.

reducir la longitud del paso h hace que la recta tangente y la falsa tangente sean más parecidas, son iguales cuando $h = 0$. De modo que la pendiente de la recta tangente se alcanza en el límite

$$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

que es la definición misma de derivada. Un detalle importante a tener en cuenta para el uso que le vamos a dar a este concepto, es que la verticalidad de la recta tangente en un punto de la gráfica de una función esta relacionada con la velocidad de cambio de la variable Y con respecto a X en ese punto especifico, quiere decir que, a cambios más bruscos corresponden rectas más verticales y a cambios más suaves corresponden rectas más horizontales.

2.3. Generalización de la derivada para dos variables

Aunque ya se ha discutido el concepto de diferenciación, para poder aplicar estos resultados a nuestro problema de interés, es necesario una ampliación del tema hacia funciones de más de una variable, puesto que como se menciona previamente, las imágenes son entendidas matemáticamente como objetos compuestos por una función de dos variables en una, se ha de pasar por el cálculo vectorial. Aunque el cálculo vectorial involucra funciones de más de de dos variables y en más de una variable, para no exceder las necesidades teóricas del tema, presento una adaptación de los conceptos generalizados, simplificados únicamente a funciones de dos variables en una. Es decir, objetos de la forma $f : U \subset R^2 \to R$.

2.3.1. Definición de límite para varias variables

Como paso previo a la definición general de límite es necesario construir una serie de objetos matemáticos que conciernen al cálculo vectorial, las siguientes definiciones son parte de la exposición que hacen (Marsden, Tromba, y Mateos, 1991).

Definición 5 *Sea $x_0 \in R^n$ y sea r un número real positivo. El **disco abierto** de radio r y centro x_0 se define como el conjunto de todos los puntos x tales que $||x - x_0|| < r$. Este conjunto se denota por $D_r(x_0)$ y es el conjunto de puntos x cuya distancia a x_0 es menor que r.*

Si nos enfocamos únicamente en el espacio que nos interesa (R^2 o el plano real) un disco abierto es exactamente el interior de un circulo con centro en el punto x_0 y de radio r. Es importante notar que la desigualdad estricta $||x - x_0|| < r$ no permite que los puntos mismos de la circunferencia sean

parte del conjunto $D_r(x_0)$. Si se esta interesado en considerar también el borde, una definición idéntica salvo en la desigualdad (que seria $||x - x_0|| \leq r$) denota dicho conjunto.

Con el concepto de disco abierto podemos entonces definir el concepto de conjunto abierto, un objeto similar pero con la libertad de no tener necesariamente forma de disco, ni estar compuesto únicamente de un solo cuerpo.

Definición 6 *Sea $U \subset R^n$ (esto es, sea U un subconjunto de R^n). Decimos que U es un* ***conjunto abierto*** *cuando para cualquier punto x_0 em U existe algun $r > 0$ tal que $D_r(x_0)$ esta contenido en U.*

Definición 7 *Designaremos como una* ***vecindad*** *de x a un conjunto abierto U que contiene al punto x. Por ejemplo, $D_r(x_0)$ es una vecindad de x_0 para cualquier $r > 0$.*

Definición 8 *Sea $A \subset R^n$. Un punto $x \in R^n$ es un* ***punto frontera*** *de A si toda vecindad de x contiene al menos un punto en A y al menos un punto fuera de A.*

Una vez establecidas las definiciones previas de conjunto abierto, vecindad y punto frontera, podemos pasar a revisar la definición de límite para funciones de varias variables.

Definición 9 *Sea $f : A \subset R^n \to R^m$, donde A es un conjunto abierto. Sea x_0 un punto en A o en la frontera de A, y sea V una vecindad de $b \in R^m$. Decimos que f esta finalmente en V conforme x tiende a x_0 si existe una vecindad U de x_0 tal que $x \neq x_0$, $x \in U$ y $x \in A$ implica $f(x) \in V$. Decimos que $f(x)$* ***tiende a b*** *cuando x tiende a x_0 y se representa por;*
$\lim_{x \to x_0} f(x) = b$ *ó* $f(x) \to b$ *cuando* $x \to x_0$
cuando, dada cualquier vecindad V de b, f esta finalmente en V conforme x tiende a x_0. Puede ser que cuando x tienda a x_0 los valores de $f(x)$ no se acerquen a un número particular. En ese caso decimos que $\lim_{x \to x_0} f(x)$ *no existe.*

Deacuerdo a nuestro interés, prestar atención al caso de funciones de dos variables en una de la forma $f : A \subset R^2 \to R$, en este caso, el conjunto abierto U vecindad de x_0 sera un objeto bidimensional (como hemos venido entendiendo el concepto de conjunto abierto) y la vecindad de $b \in R^m$ (que

ahora seria, $b \in R$ osea un punto en la recta real) referida como V puede, para fines prácticos, ser entendida como un intervalo abierto, de la forma $(b-\delta, b+\delta)$ con $\delta > 0$ arbitrario.

Con lo que, afirmar que $\lim_{x \to x_0} f(x) = b$, es asegurar la existencia de un conjunto abierto compuesto por puntos x suficientemente próximos a x_0 para que sus imágenes sean tan próximos a b como sea requerido por δ. El siguiente teorema proporcionado por (Marsden y cols., 1991) presenta una equivalencia de la definición de límite para varias variables con un enunciado que es más similar a la version para funciones de valor y variable real.

Teorema 1 *Sea $f : A \subset R^n \to R^m$ y sea x_0 un elemento de A o un punto frontera de A. Entonces $\lim_{x \to x_0} f(x) = b$ si y solo si para todo número $\epsilon > 0$ existe un $\delta > 0$ tal que para cualquier $x \in A$ que satisfaga $0 < ||x - x_0|| < \delta$, tenemos $||f(x) - b|| < \epsilon$.*

En la gráfica de la figura 2.4 se puede visualizar la manera en la que todos los elementos de U tienen una imagen dentro del intervalo centrado en b de radio δ.

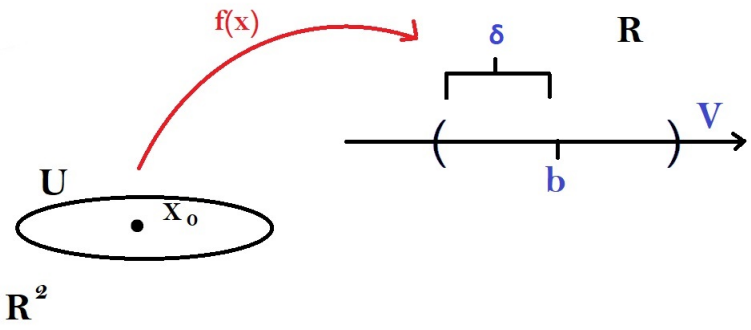

Figura 2.4: Representación gráfica del límite de una función de varias variables en una variable.

2.3.2. Definición de gradiente

Como conclusión de todo el capítulo previo, definimos el gradiente, el concepto matemático fundamental para el funcionamiento de los algoritmos de detección de bordes que vamos a estudiar. Si bien aun sera necesario realizar un ultimo paso antes de poder construir los algoritmos, entender este concepto es clave para todo lo demas.

Definición 10 *Sea $U \subset R^n$ un conjunto abierto y $f : U \subset R^n \to R$ una función con valores reales. Entonces $\partial f/\partial x_1$, ...,$\partial f/\partial x_n$ las **derivadas parciales** de f con respecto a la primera, segunda, ..., n-esima variable son las funciones con valores reales, de n variables, las cuales en el punto $(x_1, ..., x_n) = x$, están definidas por;*

$$\frac{\partial f}{\partial x_j}(x_1, ..., x_n) = \lim_{h \to 0} \frac{f(x_1, x_2, ..., x_j + h, ..., x_n) - f(x_1, ..., x_n)}{h}$$

$$= \lim_{h \to 0} \frac{f(x + he_j) - f(x)}{h}$$

si existen los limites, donde $1 \leq j \leq n$ y e_j es el j-ésimo vector de la base usual definido por $e_j = (0, ..., 1, ..., 0)$ con 1 en el j-ésimo lugar.

Esto llevado a dos dimensiones involucra a una función $f : U \subset R^2 \to R$. Lo que implica únicamente dos derivadas parciales

$$\frac{\partial f}{\partial x_1}(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

y

$$\frac{\partial f}{\partial x_2}(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1, x_2 + h) - f(x_1, x_2)}{h}$$

La poca cantidad de variables en este caso, permite diferenciar las variables x_1 y x_2 de una manera más cómoda usando x y y respectivamente. Lo que nos da las expresiones:

$$\frac{\partial f}{\partial x}(x, y) = \lim_{h \to 0} \frac{f(x + h, y) - f(x, y)}{h}$$

y

$$\frac{\partial f}{\partial y}(x, y) = \lim_{h \to 0} \frac{f(x, y + h) - f(x, y)}{h}$$

Esta ultima expresión puede ser bastante reconocible.

Un punto a considerar de suma importancia es notar que el límite

$$\lim_{h\to 0} \frac{f(x_1, x_2, ..., x_j + h, ..., x_n) - f(x_1, ..., x_n)}{h}$$

es de hecho un límite en una variable. Limitándonos al caso de dos variables para facilitar la notación pero siendo esta idea fácilmente generalizable, si consideramos a la función $f(x, y)$ de dos variables como una función de una sola variable tratando a la otra como una constante tendríamos el par de funciones $f_y(x) = f(x, c)$ y $f_x(y) = f(c, x)$ de variable y valor real. Ahora observe como en la expresión

$$\lim_{h\to 0} \frac{f(x + h, y) - f(x, y)}{h}$$

La variable y se mantiene constante (ocurre lo mismo con x en la otra expresión), de modo que podemos escribir esta expresión como

$$\lim_{h\to 0} \frac{f_y(x + h) - f_y(x)}{h}$$

Situación en la que basta la definición de límite en una sola variable, y si se es observador, puede verse la definición de derivada en una sola variable también.

Definición 11 *Considerando el caso especial $f : U \subset R^n \to R$. La matriz de $1xn$:*

$$Df(x) = [\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}]$$

*Formada por el correspondiente vector $[\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}]$, es llamada el **gradiente** de f y se denota por $gradf$ o $\bigtriangledown f$.*

Finalmente una vez establecida la definición de derivada parcial, y para el caso especifico de dos variables, el gradiente sera el vector

$$\bigtriangledown f(x, y) = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

de derivadas parciales.

2.3.3. Interpretación geométrica del gradiente

Observe que las derivadas parciales parten de considerar a las funciones antes descritas f_x y f_y, note que el dominio de estas funciones pueden ser entendidos como los conjuntos $Domf_x := \{(x,y) \in R^2 | x = k_x\}$ y $Domf_y := \{(x,y) \in R^2 | x = k_y\}$, para k_x,$k_y \in R$ arbitrarias pero fijas, que corresponderían a lineas rectas paralelas al eje Y y X, que pasan por $(k_x, 0)$ y $(0, k_y)$ respectivamente. La funciónes son entonces un recorrido a la función principal $(f(x,y))$ pero únicamente a lo largo de estas rectas.

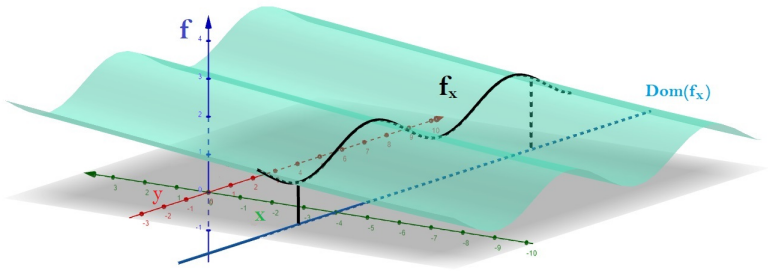

Figura 2.5: Representación gráfica del cálculo del gradiente en una función de dos variables en una.

De esta manera es más sencillo imaginar como las derivadas parciales nos proporcionan información del crecimiento de la función principal en una dirección en particular, ya sea la del eje X o la del eje Y. De modo que $\frac{\partial f}{\partial x}$ es la tasa de crecimiento de la función f en dirección al vector $(1,0)$, mientras que $\frac{\partial f}{\partial y}$ lo es en la dirección de $(0,1)$. Así el vector $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = \frac{\partial f}{\partial x}(1,0) + \frac{\partial f}{\partial y}(0,1)$ va a representar la dirección en la que la función crece con mayor rapidez.
Un entendimiento más amplio y profundo de la diferenciación multivariable y el cálculo vectorial en general puede encontrarse en (Marsden y cols., 1991).

Capítulo 3

Máscaras para detectar bordes

3.1. Series de Taylor para aproximar la derivada de forma numérica

Hay que recordar que las limitaciones tecnológicas no hacen posible almacenar una imagen en memoria de manera perfecta debido a la naturaleza analógica de la luz y del espacio, por lo que los mecanismos que nos permiten almacenar y posteriormente reconstruir una imagen de la manera más fiel a la realidad posible se basan en la discretización de la misma, esto hace que el procesamiento digital de una imagen se haga sobre representaciones discretas de las funciones de las imágenes, haciendo necesario recurrir a una herramienta matemática que nos permita determinar la derivada de manera discreta.

3.1.1. Definición de polinomio de Taylor

Comencemos por observar el comportamiento de las derivadas de un polinomio de orden n, digamos:

$$p(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$$

y observe primero que $p(0) = a_0$. Posteriormente, al derivar $p(x)$ para obtener la expresión $p'(x) = a_1 + 2a_2x + ... + na_nx^{n-1}$, se tendrá que $p'(0) = 2a_1$. Calcular $p''(0), p^3(0), ..., p^n(0)$ bastara para convencerse de que:

$$p^k(0) = k!a_k$$

De donde se sigue que

$$a_k = \frac{p^k(0)}{k!}$$

Siempre que $0! = 1$ y $p^0(x) = p(x)$. Es decir, los coeficientes de un polinomio pueden calcularse a partir de las derivadas del mismo. Si vamos más lejos, y consideramos el polinomio centrado en $x - a$, esto es:

$$p(x) = a_0 + a_1(x-a) + a_2(x-a)^2 + ... + a_n(x-a)^n$$

No es difícil llegar a la conclusión de que los coeficientes estarán determinados por la expresión

$$a_k = \frac{p^k(a)}{k!}$$

Definición 12 *Si ahora consideramos una función que no sea necesariamente un polinomio, cuyas primeras n derivadas existen en un punto a de su dominio, el polinomio*

$$P_{n,a}(x) = a_0 + a_1(x-a) + a_2(x-a)^2 + ... + a_n(x-a)^n$$

Donde $a_k = \frac{f^k(a)}{k!}$ *con* $0 \leq k \leq n$.

Recibe el nombre de ***Polinomio de Taylor de grado*** n ***para*** f ***en*** a *(Spivak, 1988).*

(Spivak, 1988) también nos presenta y demuestra el siguiente teorema;

Teorema 2 *Supongamos que* f *es una función para la cual*

$$f'(a), ..., f^{(n)}(a)$$

existen todas. Sea

$$a_k = \frac{f^{(k)}(a)}{k!}, a \leq k \leq n,$$

y definamos

$$P_{n,a}(x) = a_0 + a_1(x-a) + ... + a_n(x-a)^n$$

Entonces

$$\lim_{x \to a} \frac{f(x) - P_{n,a}(x)}{(x-a)^n} = 0$$

Elementalmente este teorema nos garantiza que conforme $x \to a$, el polinomio de Taylor se aproxima a la función f más rápidamente de lo que lo hace x hacia a.
Note también que eso sugiere que mientras un polinomio de grado 1 se aproxima a f más rápido de lo que $x - a$ se aproxima a 0. Un polinomio de grado 2 lo hará más rápido que $(x-a)^2$ que de hecho lo hace más rápido que $(x-a)$. Esto ultimo nos permite inferir la existencia de un concepto de 'orden de igualdad'.

Definición 13 *Dos funciones f y g se dice que son **iguales hasta en orden** n **en** a si*

$$\lim_{x \to a} \frac{f(x) - g(x)}{(x-a)^n} = 0$$

Con ello se puede dar paso a la duda de si una función f es igual en un orden dado n y en un punto dado a a un unico polinomio, y la respuesta es que si, deacuerdo al teorema a continuación.

Teorema 3 *Sean P y Q dos polinomios en $(x-a)$ de grado $\leq n$, y supongamos que P y Q son iguales hasta el orden n en a. Entonces $P = Q$.*

Esto ultimo nos garantiza que si dos polinomios P y Q son iguales hasta en orden n en a a una función f, entonces $\lim_{x \to a} \frac{P(x)-f(x)}{(x-a)^n} = 0$ y $\lim_{x \to a} \frac{f(x)-Q(x)}{(x-a)^n} = 0$. Luego

$$\lim_{x \to a} \frac{P(x) - Q(x)}{(x-a)^n} = \lim_{x \to a} \frac{P(x) - f(x) + f(x) - Q(x)}{(x-a)^n} =$$

$$\lim_{x \to a} \frac{P(x) - f(x)}{(x-a)^n} + \lim_{x \to a} \frac{f(x) - Q(x)}{(x-a)^n} = 0$$

Es decir, ambos polinomios son iguales en a hasta en orden n, de donde se sigue que son el mismo polinomio.

Una vez establecido el polinomio de Taylor y habiendo verificado que este es único, podemos comenzar a explorar el error de aproximación del mismo. Comenzando por definir el concepto de resto

Definición 14 *Si f es una función para la cual existe $P_{n,a}(x)$, definimos el **resto** $R_{n,a}(x)$ por*

$$f(x) = P_{n,a}(x) = P_{n,a}(x) + R_{n,a}(x)$$

$$= f(a) + f'(a)(x-a) + ... + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_{n,a}(x)$$

Finalmente, el teorema de Taylor que a continuación se enuncia presenta diversas expresiones que permiten estimar la magnitud de el resto.

Teorema 4 *Suponga que* $f', ..., f^{(n+1)}$ *están definidas sobre* $[a.x]$*, y que* $R_{n,a}(x)$ *esta definido por*

$$f(x) = f(a) + f'(a)(x-a) + ... + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_{n,a}(x)$$

Entonces

$$(1)\ R_{n,a}(x) = \frac{f^{(n+1)}(t)}{n!}(x-t)^n(x-a)\ \textit{para algun } t \textit{ de } (a,x)$$

$$(2)\ R_{n,a}(x) = \frac{f^{(n+1)}(t)}{(n+1)!}(x-a)^{n+1}\ \textit{para algun } t \textit{ de } (a,x)$$

Ademas, si $f^{(n+1)}$ *es integrable sobre* $[a,x]$*, entonces*

$$(3)\ R_{n,a}(x) = \int_a^x \frac{f^{(n+1)}(t)}{n!}(x-t)^n dt$$

(Spivak, 1988)

3.1.2. Aproximación numérica de la derivada

De acuerdo a como presentan (Ledesma, Rosas, y Bernal, 2015), en el anexo de su obra. Partimos de la ecuación ya bien conocida del polinomio de Taylor para una función.

$$f(x) = f(a) + \frac{(x-a)}{1!}f'(a) + \frac{(x-a)^2}{2!}f''(a) + ... + \frac{(x-a)^n}{n!}f^{(n)}(a) + R_{n,a}(x)$$

A partir de esta definición, seguimos como sugieren (Ledesma y cols., 2015) y consideramos la ecuación para $n = 1$ y $x = a + \Delta x$.

$$f(a+\Delta x) = f(a) + \frac{\Delta x}{1} f'(a) + R_{1,a}(a+\Delta x)$$

Despejando $f'(a)$ tenemos

$$f'(a) = \frac{f(a+\Delta x) - f(a)}{\Delta x} - R_{1,a}(a+\Delta x)$$

Finalmente, teniendo en cuenta que $R_{1,a}(a+\Delta x)$ escapa de los alcances del cálculo numérico, podemos considerar

$$f'(a) = \frac{f(a+\Delta x) - f(a)}{\Delta x}$$

Como nuestra aproximación numérica para la derivada también llamada **diferencia progresiva**.

Junto con la expresión anterior, si tomamos $n = 1$ y $x = a - \Delta x$, un procedimiento análogo nos llevara a la expresión

$$f'(a) = \frac{f(a) - f(a-\Delta x)}{\Delta x}$$

que es conocida como la **diferenciación regresiva**. Ahora bien , si en cambio consideramos $n = 2$ y $x = a + \Delta x$;

$$f(a+\Delta x) = f(a) + \frac{\Delta x}{1} f'(a) + \frac{\Delta x^2}{2!} f''(a) + R_{2,a}(a+\Delta x)$$

en conjunto con la ecuación para $x = a - \Delta x$

$$f(a-\Delta x) = f(a) - \frac{\Delta x}{1} f'(a) + \frac{\Delta x^2}{2!} f''(a) + R_{2,a}(a-\Delta x)$$

y restamos ambas expresiones, se tendrá que

$$f(a+\Delta x) - f(a-\Delta x) = 2\Delta x f'(a) + R_{2,a}(a+\Delta x) - R_{2,a}(a-\Delta x)$$

de donde podemos obtener

$$f'(a) = \frac{f(a+\Delta x) - f(a-\Delta x)}{2\Delta x} - R_{2,a}(a+\Delta x) + R_{2,a}(a-\Delta x)$$

de donde se sigue la llamada **diferencia centrada**;

$$f'(a) = \frac{f(a + \Delta x) - f(a - \Delta x)}{2\Delta x}$$

que requiere un punto adicional para calcular la derivada, pero de acuerdo con (Ledesma y cols., 2015) tiene una mayor precision.

3.1.3. Aproximación numérica de la segunda derivada

Nuevamente, partimos del desarrollo de Taylor

$$f(a + \Delta x) = f(a) + \frac{\Delta x}{1} f'(a) + \frac{\Delta x^2}{2!} f''(a) + \frac{\Delta x^3}{3!} f'''(a) + R_{3,a}(a + \Delta x)$$

y

$$f(a - \Delta x) = f(a) - \frac{\Delta x}{1} f'(a) + \frac{\Delta x^2}{2!} f''(a) - \frac{\Delta x^3}{3!} f'''(a) + R_{3,a}(a - \Delta x)$$

Si sumamos ambas ecuaciones se tiene que

$$f(a + \Delta x) + f(a - \Delta x) = 2f(a) + 2\frac{\Delta x^2}{2!} f''(a) + R_{3,a}(a + \Delta x) + R_{3,a}(a - \Delta x)$$

Despejando $f''(a)$

$$\frac{f(a - \Delta x) - 2f(a) + f(a + \Delta x)}{\Delta x^2} - R_{3,a}(a + \Delta x) - R_{3,a}(a - \Delta x) = f''(a)$$

Así, nuestra aproximación numérica a la segunda derivada, despreciando el error de truncamiento $R_{3,a}(a + \Delta x) + R_{3,a}(a - \Delta x)$ es;

$$\frac{f(a - \Delta x) - 2f(a) + f(a + \Delta x)}{\Delta x^2} = f''(a)$$

Si se desea hacer uso de más puntos (Ledesma y cols., 2015) presentan algunas expresiones adicionales.

3.1.4. Derivadas en dos dimensiones

Recordando la manera en la que entendemos las imágenes digitales, como funciones de dos variables discretizadas en una malla de niveles de gris que podemos almacenar como una matriz, tiene sentido preguntarnos por como encontrar aproximaciones a la derivada para este tipo de funciones. (Ledesma y cols., 2015) disponen de una manera de aproximar el gradiente en un punto de una cuadricula de valores, a partir de una generalización del método que ya tenemos basado en series de Taylor para las funciones en una variable.

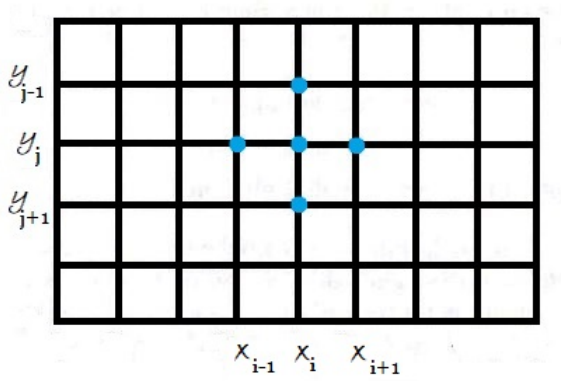

Figura 3.1: Representación de la cuadricula de valores de una función discretizada a la que se le desea aproximar derivada.
(Ledesma y cols., 2015)

A partir del teorema de Taylor para funciones de dos variables (x, y) podemos escribir para un punto en particular (x_i, y_j)

$$f(x_i + \Delta x, y_j) = f(x_i, y_j) + \Delta x \frac{\partial f(x_i, y_j)}{\partial x} + R_{1,(x_i,y_j)}(x_i + \Delta x, y_j)$$

$$f(x_i, y_j + \Delta y) = f(x_i, y_j) + \Delta y \frac{\partial f(x_i, y_j)}{\partial y} + R_{1,(x_i,y_j)}(x_i, y_j + \Delta y)$$

Así, la **aproximación en diferencias finitas hacia adelante** de $\frac{\partial f}{\partial x}$ y $\frac{\partial f}{\partial y}$ serán

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i, y_j)}{\Delta x}$$

y

$$\frac{\partial f(x_i, y_j)}{\partial y} = \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j)}{\Delta y}$$

La aproximación en **diferencias hacia atrás** son

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i, y_j) - f(x_i - \Delta x, y_j)}{\Delta x}$$

y

$$\frac{\partial f(x_i, y_j)}{\partial y} = \frac{f(x_i, y_j) - f(x_i, y_j - \Delta y)}{\Delta y}$$

y la aproximación en **diferencias centradas**

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{2\Delta x}$$

y

$$\frac{\partial f(x_i, y_j)}{\partial y} = \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j - \Delta y)}{2\Delta y}$$

Por ultimo, es posible calcular de manera análoga al caso unidimensional $\frac{\partial^2 f}{\partial x^2}$ y $\frac{\partial^2 f}{\partial y^2}$

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x_i - \Delta x, y_j) - 2f(x_i, y_j) + f(x_i + \Delta x, y_j)}{\Delta x^2}$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{f(x_i, y_j - \Delta y) - 2f(x_i, y_j) + f(x_i, y_j + \Delta y)}{\Delta y^2}$$

que es la aproximación en **diferencias centradas** del **Laplaciano** que [(Sucar y Gómez, 2011) define como sigue.

Definición 15 *El **Laplaciano** de una función de dos variables se define como*

$$\bigtriangledown f = (\frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2})$$

3.2. Detección de bordes

"Viendo la imagen como una función, un contorno implica una discontinuidad en dicha función, es decir donde la función tiene un valor de gradiente o derivada alta"(Sucar y Gómez, 2011).

Con la cita anterior como definición, podemos comenzar a desarrollar el mecanismo por el cual vamos a detectar los bordes de los objetos en una imagen digital. Si apreciamos con detenimiento los bordes en una imagen notaremos que estos son variaciones en el nivel de intensidad.

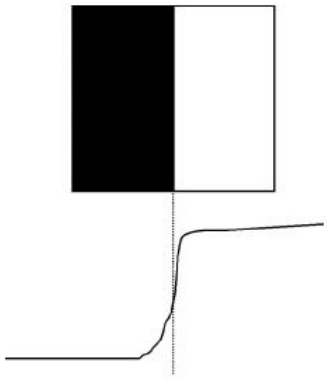

Figura 3.2: Visualización de la función cuando ocurre un cambio brusco en el nivel de gris.
(Sucar y Gómez, 2011)

de modo que, si calculamos (o aproximamos) la derivada en cada punto de una imagen y después comparamos ese valor con un cierto umbral, podemos determinar si el punto o pixel es parte de un borde. Es común también hacer uso de la magnitud del vector gradiente $[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$ que no es más que la norma del vector, que una vez conocidas las derivadas parciales, puede calcularse como:

$$||[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]|| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$$

3.2.1. Operadores de Roberts

La manera más directa de aproximar la derivada es con una diferencia hacia adelante, dado un pixel (x_i, y_j), partiendo del par de ecuaciones

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i, y_j)}{\Delta x}$$

y

$$\frac{\partial f(x_i, y_j)}{\partial y} = \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j)}{\Delta y}$$

En nuestra cuadricula de datos, si hacemos $\Delta x = 1 = \Delta y$ entonces $x_i + \Delta x = x_{i+1}$ y $y_j + \Delta y = y_{j+1}$. De modo que las derivadas quedan como sigue

$$\frac{\partial f(x_i, y_j)}{\partial x} = f(x_{i+1}, y_j) - f(x_i, y_j)$$

$$\frac{\partial f(x_i, y_j)}{\partial y} = f(x_i, y_{j+1}) - f(x_i, y_j)$$

Podemos comenzar a imaginar como implementar una mascara de convolución que se encargue de hacer estos cálculos. Note que nuestra imagen puede entenderse como la matriz;

$$\begin{bmatrix} f(x_1, y_1) & f(x_2, y_1) & \cdots & f(x_N, y_1) \\ f(x_1, y_2) & f(x_2; y_2) & \cdots & f(x_N, y_2) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_1, y_M) & f(x_2, y_M) & \cdots & f(x_N, y_M) \end{bmatrix}$$

Entonces, un par de máscaras que permitirán calcular $f(x_{i+1}, y_j) - f(x_i, y_j)$ y $f(x_i, y_{j+1}) - f(x_i, y_j)$ serian;

$$\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$$

y

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$$

que calculan respectivamente $\frac{\partial f}{\partial x}$ y $\frac{\partial f}{\partial y}$ para el pixel (x_i, y_j) coincidiendo en la posición superior izquierda de la mascara. Esto es puesto que $\sum_{i,j=1}^{2N+1} c_{i,j} =$

0 y por ende se toma $c = 1$.

No es difícil darse cuenta que para las diferencias regresivas $f(x_i, y_j) - f(x_{i-1}, y_j)$ y $f(x_i, y_j) - f(x_i, y_{j-1})$ es necesario centrarse en el pixel inferior derecho y usar las matrices

$$\begin{bmatrix} 0 & 0 \\ -1 & 1 \end{bmatrix}$$

y

$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}$$

Ademas, (Sucar y Gómez, 2011) hacen uso de las diferencias cruzadas para obtener $\frac{\partial f}{\partial x}$ y $\frac{\partial f}{\partial y}$ mediante las respectivas matrices

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

y

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Este ultimo par de máscaras de convolución son llamados **Operadores de Roberts**.

Es verdad que la convolución esta específicamente definida para matrices de orden impar, por lo que puede resultar un poco forzado el hacer que matrices de orden dos funcionen como máscaras, el primer problema en aparecer es la ambigüedad de determinar cual es el pixel central de la matriz. En el próximo capítulo se exploran algunas soluciones a estos inconvenientes, aunque afortunadamente, los operadores de Roberts no son muy utilizados.

3.2.2. Operadores de Prewitt

Los operadores Prewitt a diferencia de de los de Roberts, si consideran una mascara de convolución de grado impar, de hecho, se trata de matrices de $3x3$, en las que, el centro de la matriz, a veces llamado kernel, esta asociado con el punto (x_i, y_j) del que queremos averiguar la derivada.

$$\begin{bmatrix} f(x_{i-1}, y_{j-1}) & f(x_i, y_{j-1}) & f(x_{i+1}, y_{j-1}) \\ f(x_{i-1}, y_j) & f(x_i, y_j) & f(x_{i+1}, y_j) \\ f(x_{i-1}, y_{j+1}) & f(x_i, y_{j+1}) & f(x_{i+1}, y_{j+1}) \end{bmatrix}$$

Teniendo toda esta vecindad de pixeles al rededor del pixel de interés, es natural desear hacer uso de la ecuación de **diferencias centradas** que hace uso de un punto posterior y uno previo, algo que ya nos proporciona una matriz $3x3$.

Sin embargo, las ecuaciones

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{2\Delta x} = \frac{f(x_{i+1}, y_j) - f(x_{i-1}, y_j)}{2}$$

$$\frac{\partial f(x_i, y_j)}{\partial y} = \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j - \Delta y)}{2\Delta y} = \frac{f(x_i, y_{j+1}) - f(x_i, y_{j-1})}{2}$$

Requieren que las diferencias se dividan entre 2, algo que no es tan fácil de hacer con convolución, la solución será calcular $2\frac{\partial f}{\partial x}$ y $2\frac{\partial f}{\partial y}$, posteriormente podemos duplicar el valor del umbral para obtener el mismo resultado.

Así, la manera de calcular $f(x_{i+1}, y_j) - f(x_{i-1}, y_j)$ y $f(x_i, y_{j+1}) - f(x_i, y_{j-1})$ mediante una matriz $3x3$ será con las máscaras

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

y

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Si bien estas matrices son máscaras de convolución perfectamente funcionales, se puede apreciar que se esta desperdiciando la información que proporcionan un par de columnas (o filas) plagadas de cero, podemos darles el mismo uso que la columna (o fila) central y así tener una idea de la localidad del pixel central.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

y

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Son los conocidos **Operadores de Prewitt**, obtiene cada uno tres derivadas parciales que aparecerán sumadas, aunque este detalle también puede tratarse en el paso de la binarización.

3.2.3. Operadores de Sobel

Deacuerdo con (Sucar y Gómez, 2011) tanto Roberts como Prewitt tienen el inconveniente de ser sensibles al ruido, hay toda una rama del procesamiento de imágenes dedicada al desarrollo de filtros que se encargan de suavizar las imágenes para limpiar este ruido.

Sin embargo para evitar desvíos innecesarios simplemente se hablara brevemente del filtro Gaussiano; Considerando una media igual a cero, la función de transformación de un filtro tipo Gaussiano es:

$$T(x, y) = e^{-[(x^2+y^2)/2\pi\sigma^2]}$$

Donde σ es la desviación estándar. Para una mascara de 3x3 los valores de un **filtro Gaussiano** típico son

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 8 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

La cantidad de "suavizamiento"que realiza el filtro Gaussiano se puede controlar variando la desviación estándar y el tamaño de la mascara (Sucar y Gómez, 2011).

Elementalmente, el filtro de suavizamiento hace el cálculo

$$f(x_{i-1}, y_{j-1}) + \frac{2}{10} f(x_i, y_{j-1}) + f(x_{i+1}, y_{j-1}) + \frac{2}{20} f(x_{i-1}, y_j) +$$

$$\frac{8}{20} f(x_i, y_j) + \frac{2}{20} f(x_{i+1}, y_j) + f(x_{i-1}, y_{j+1}) + \frac{2}{20} f(x_i, y_{j+1}) + f(x_{i+1}, y_{j+1})$$

Que es una suma ponderada de los pixeles que conforman la vecindad de (x_i, y_j) para hacer que la vecindad se vuelva más uniforme y elimine el ruido. Los **operadores de Sobel** entonces parten de los de Prewitt, adicionando ciertos pesos en la mascara que aproximen a un suavizamiento Gaussiano (Sucar y Gómez, 2011), esta modificación da lugar a las máscaras.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

3.2.4. Operador Laplaciano

'A finales de los 70s. David Marr estudio la visión de los mamíferos e ideo una teoría que integraba prácticamente todo lo que se conocía sobre la visión biológica. Su detector de orillas se basa en segundas derivadas o Laplaciano de una Gaussiana' (Sucar y Gómez, 2011).

Motivados por la cita anterior, vamos a construir un filtro que use las segundas derivadas parciales, recuérdese que antes definimos el Laplaciano como el vector de las segundas derivadas parciales de una función. Así mismo también habíamos encontrado una aproximación numérica, la cual, aplicada a nuestra imagen digital tiene la forma

$$\frac{\partial^2 f}{\partial x^2} = f(x_{i-1}, y_j) - 2f(x_i, y_j) + f(x_{i+1}, y_j)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x_i, y_{j-1}) - 2f(x_i, y_j) + f(x_i, y_{j+1})$$

Casi inmediatamente podemos deducir un par de máscaras que podrían encargarse de este cálculo

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

y

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

La verdadera razón por la que se recurre a la segunda derivada es que proporciona una mejor precision para estimar la localización de la orilla, será exactamente donde la segunda derivada cruza con cero.

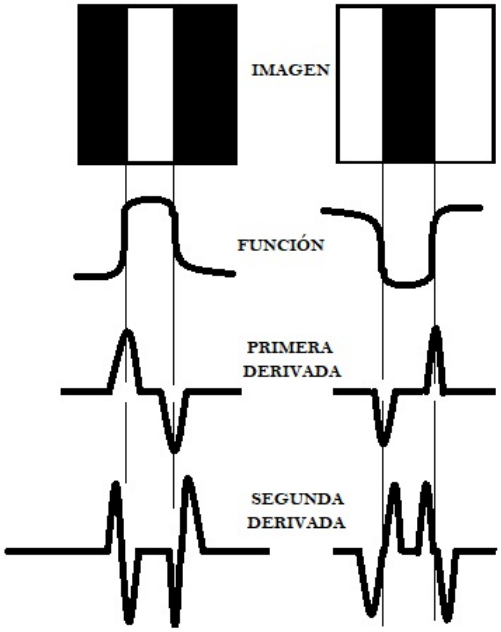

Figura 3.3: Ejemplo de como se ven las gráficas de la función, su derivada y su segunda derivada según hay cambios de intensidad en la imagen.
(Sucar y Gómez, 2011)

Hasta ahora, las derivadas parciales se han calculado por separado. cada una con su mascara de convolución, pero esto tiene el inconveniente de que los bordes que están orientados en la dirección paralela a la derivada no serán detectados, es necesario pues, aplicar ambos filtros y combinarlos en un proceso posterior o, combinar ambos procedimientos derivativos en un solo filtro.

La segunda es la solución que se obtiene al sumar ambas máscaras de convolución, para el caso particular en el que estamos (Operador Laplaciano) obtendremos la matriz

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Observe que ahora, el filtro estará calculando $\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$, los valores obtenidos serán más grandes que con las matrices por separado, lo que puede requerir ajustes al umbral, aunque para este caso en especifico no es necesario debido a la naturaleza de la segunda derivada.

Una ultima modificación que suele hacerse al filtro es invertir el signo de sus valores, esto no tiene una repercusión importante pues los signos de los pixeles pos-filtrado simplemente se invertirán, lo cual no dará problemas pues únicamente estamos interesados en los cambios de signo.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Esta ultima es entonces, la mascara de $3x3$ para el **operador Laplaciano**. (Sucar y Gómez, 2011) mencionan que el principal defecto de este filtro es la generación de orillas dobles, esto tiene sentido pues en los puntos en los que la primera derivada presenta un pico, la segunda derivada presentara dos, uno positivo y uno negativo ambos interpretados como puntos de borde.

3.2.5. Operadores de Kirsch

Los métodos de gradiente, usualmente hacen únicamente uso de las derivadas parciales, pero hay un concepto poco usado en el cálculo vectorial

llamado **derivada direccional**. En (Marsden y cols., 1991) se proporciona una definición

Definición 16 *Si $f : R^2 \to R$, la **derivada direccional** de f en x en la dirección de un vector v está dada por*

$$\frac{d}{dt} f(x + tv) \mid_{t=0}$$

si es que existe.

La derivada direccional también se puede definir por la formula

$$\lim_{h \to 0} \frac{f(x + tv) - f(x)}{h}$$

A groso modo, la derivada direccional mide la razón de cambio de una función vectorial en una dirección especifica, si esta dirección coincide con la de $(1, 0)$ o $(0, 1)$ tendremos $\frac{\partial f}{\partial x}$ y $\frac{\partial f}{\partial y}$ (las derivadas parciales de siempre). Cuando este no es el caso tendremos un nuevo tipo de derivada.

Bajo esta idea se construyen los **operadores de Kirsch**, para cuatro direcciones; 0, 45, 90 y 135 grados. Estos se pueden definir para diferentes tamaño de matriz, las máscaras de $3x3$ son las siguientes.

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

Es usual que se pasen los cuatro operadores, y se escoja el valor más grande de entre todos para pasarlo por el umbral. De esta manera se puede asegurar detectar la mayor cantidad de bordes posible. (Sucar y Gómez, 2011)

Capítulo 4

Metodología

Aunque existen una gran cantidad de lenguajes de programación con librerías ya establecidas para esta clase de tareas, se decide realizar la implementacion en el lenguaje C, debido a la gran velocidad que permite, así como la posibilidad de ejecutarlo en diferentes plataformas. Ademas de que programar desde cero las funcionalidades de convolución, binarización y creación de archivos BMP proporciona una mejor comprensión del tema.

4.1. Implementación de la convolución en lenguaje C

Si bien existen, y se usan con regularidad máscaras de convolución de mayor tamaño que $3x3$, como todas las matrices que se presentan en este documento son de ese tamaño el script fue programado únicamente con la funcionalidad de matrices de orden 3, aunque no es imposible adaptarlo a matrices de mayor tamaño. Así mismo las matrices de orden 3 son suficientes para apreciar los efectos de los filtros.

Por otro lado, aunque es posible estudiar cada plano de color (RGB) de una imagen a color, lo usual es convertir la imagen a escala de grises para posteriormente procesarla, esto se consigue tomando el valor de cada nivel de color de cada plano para calcular su promedio, posteriormente colocar ese promedio en los 3 planos del archivo (BMP), haciendo esto para cada pixel, tendremos una imagen en escala de grises bastante fiel, con el único inconveniente de que cada pixel tiene tres datos repetidos en el sistema del

archivo BMP, este desperdicio de memoria es un mal menor con el que podemos convivir, ya que ahorrara el paso de almacenar la imagen en un formato distinto que sea difícil de leer.

4.1.1. Descripción del algoritmo

El script en C consiste en una serie de funciones que, ademas de los valores particulares que necesitan para trabajar, reciben como argumento dos cadenas de texto la primera debe contener el nombre y dirección de un archivo en formato BMP que se desee procesar, y la segunda el nombre y dirección del archivo (también BMP) que será creado posterior al procesamiento. Un ejemplo de ello es la función:

```
int filtro_gris(char nombre[],char out[]);
```

que convierte una imagen a escala de grises, utiliza dos argumentos únicamente, el archivo de entrada y el de salida, descritos anteriormente. Esta función existe para poder preparar cualquier imagen antes de aplicar cualquier tipo de filtro. La siguiente función en orden de complejidad es la función

```
int umbralizar(char nombre[],char out[], int umbral);
```

Que ademas de los dos parámetros de nombre, necesita que se especifique un valor de umbral, los pixeles que se encuentren por debajo de ese umbral serán convertidos en blanco, mientras que el resto serán convertidos a negro, dando lugar a una imagen binarizada, esta función es una parte importante del algoritmo de detección de bordes ya que permite eliminar algo de ruido a las imágenes procesadas.

Por otro lado para poder aplicar los filtros con suficiente control, se han construido tres funciones de convolución:

```
int convolucionar(char nombre[],char out[], int matrix
    [][3]);

int convolucionar_abs(char nombre[],char out[],int
    matrix[][3]);

int convolucionar_Magnitud(char nombre[],char out[],
                           int matrix[][3], int matrix2
                              [][3]);
```

La función **convolucionar** dispone de la manera más pura de convolución, aplica la mascara sin ningún paso adicional, sus primeros argumentos son los archivos de entrada y salida, ademas de una matriz de orden 3 con la cual se aplicara la convolución, existe como punto de partida para las funciones especializadas que apliquen filtros específicos de detección de bordes, pero también permite experimentar con máscaras de convolución ajenos a estos.

Una complicación al momento de almacenar los datos en un BMP, es que estos están limitados al formato de color de 256 niveles, los cálculos de diferenciación a menudo producirán valores negativos, si estos se dejan como tal, el sistema va a remplazarlos por su complemento es decir, un valor de derivada negativa como -235 será tratado como $255 - 235 = 20$, un color muy cercano al negro, pero ademas un valor negativo cercano al cero como -3 será tratado como $255 - 3 = 252$ dando lugar a un pixel casi blanco que se confundirá con un borde donde claramente no lo hay, pues la derivada es muy cercana a cero. Ocasionando que un claro pixel de borde vea reducido su valor y un claro pixel plano halla sido realzado, lo que produce resultados inquietantes como los de la figura 4.1.

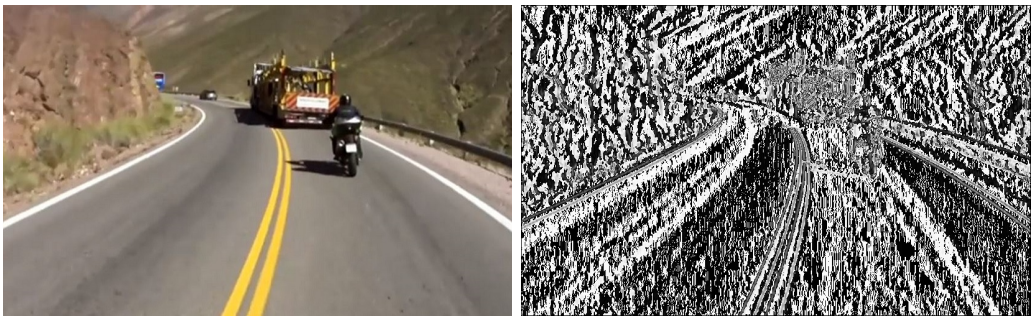

Figura 4.1: A la izquierda la imagen original, a la derecha la imagen procesada por el filtro sin hacer ningún paso extra.

La solución inmediata es añadir un paso más al proceso antes de siquiera cerrar la escritura del archivo BMP, y es simplemente calcular el valor absoluto de todos los pixeles, ese es el propósito de la función **convolucionar_abs**, con ella las derivadas grandes se mantienen grandes, lo que nos permite, aun sin el paso de la binarización, apreciar los bordes de la imagen procesada,

vea la figura 4.2.

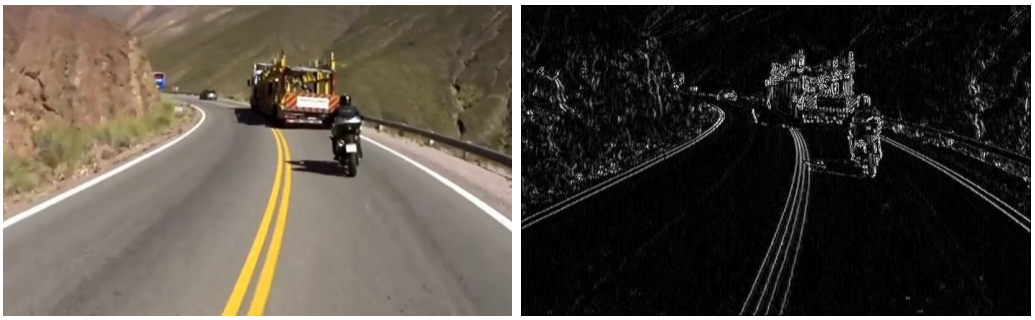

Figura 4.2: A la izquierda imagen original, a la derecha imagen procesada seguido de el cálculo del valor absoluto antes de ser guardada en el archivo.

Por ultimo la función **convolucionar_Magnitud** recibe dos matrices como argumento, ademas de los obvios input y ouput, pues tiene como objetivo convolucionar dos matrices distintas y hacer el cálculo $\sqrt{C_1^2 + C_2^2}$. Esto con el objetivo obvio de calcular la magnitud del gradiente, compuesto por dos aproximaciones a la derivada parcial. La utilidad de esta función será descrita más adelante.

Esta configuración permite poder ejecutar desde la función principal del script todo tipo de filtros de convolución, por ejemplo

```
1     filtro_gris("imagen.bmp","ImagenGris.bmp");
2
3     int prewitt[3][3]={{-1,0,1},
4                        {-1,0,1},
5                        {-1,0,1}};
6
7     convolucionar("ImagenGris.bmp","ImagenGrisLaplaciano
         .bmp",prewitt);
```

produce el resultado de la figura 4.1. Como dato adicional, si se desea calcular $\sqrt{x^2}$ en vez del valor absoluto para convoluciones que involucren a una sola matriz puede pasarse una matriz nula como cuarto parámetro de la función **convolucionar_Magnitud**.

4.1.2. Variaciones en el umbral de la binarización

Si bien la función **umbralizar** ha sido descrita, es necesario hacer hincapié en su utilidad dentro del algoritmo antes de seguir. Desde la construcción de las matrices de convolución, ya se ha mencionado cual era el papel de una función de este tipo, separar las derivadas que considerábamos significativas de las que no. También, si se lee con detenimiento la descripción en la sección previa, se percatara de que los valores altos serán caracterizados por un color negro, y los valores bajos por un color blanco, es decir, la función invierte el sentido de las intesidades del sistema BMP, que es negro para valores bajos y blanco para altos, la razón, hacer más cómodo para el ojo humano apreciar los resultados del procesamiento.

Se puede, por ejemplo, generar una secuencia de procesamiento con las siguientes lineas:

```
1   filtro_gris("img.bmp","Gris.bmp");
2
3   int horizontal[3][3]={{-1,0,1},
4                         {-1,0,1},
5                         {-1,0,1}};
6
7   int vertical[3][3]={{-1,-1,-1},
8                       { 0, 0, 0},
9                       { 1, 1, 1}};
10
11  convolucionar_Magnitud("Gris.bmp","Bordes.bmp",
      vertical,horizontal);
```

Figura 4.3: Procesado de fotografía digital.

Note como aún sin el proceso de binarización se aprecia en color blanco los bordes resultados del procesamiento; sin embargo, hay diferentes niveles de blanco, algunos más oscuros, de cierta manera, ensucian la imagen. Determinar cúal es el umbral adecuado dependerá de la imagen a procesar, por lo que se hace uso de varios procesos de binarización con diferentes umbrales hasta hallar uno que se sea conveniente para el uso que se dé.

```
1     umbralizar("Bordes.bmp","Bin10.bmp",10);
2     umbralizar("Bordes.bmp","Bin20.bmp",20);
3     umbralizar("Bordes.bmp","Bin30.bmp",30);
4     umbralizar("Bordes.bmp","Bin40.bmp",40);
5     umbralizar("Bordes.bmp","Bin50.bmp",50);
6     umbralizar("Bordes.bmp","Bin60.bmp",60);
7     umbralizar("Bordes.bmp","Bin70.bmp",70);
8     umbralizar("Bordes.bmp","Bin80.bmp",80);
9     umbralizar("Bordes.bmp","Bin90.bmp",90);
10    umbralizar("Bordes.bmp","Bin100.bmp",100);
11    umbralizar("Bordes.bmp","Bin110.bmp",110);
12    umbralizar("Bordes.bmp","Bin120.bmp",120);
```

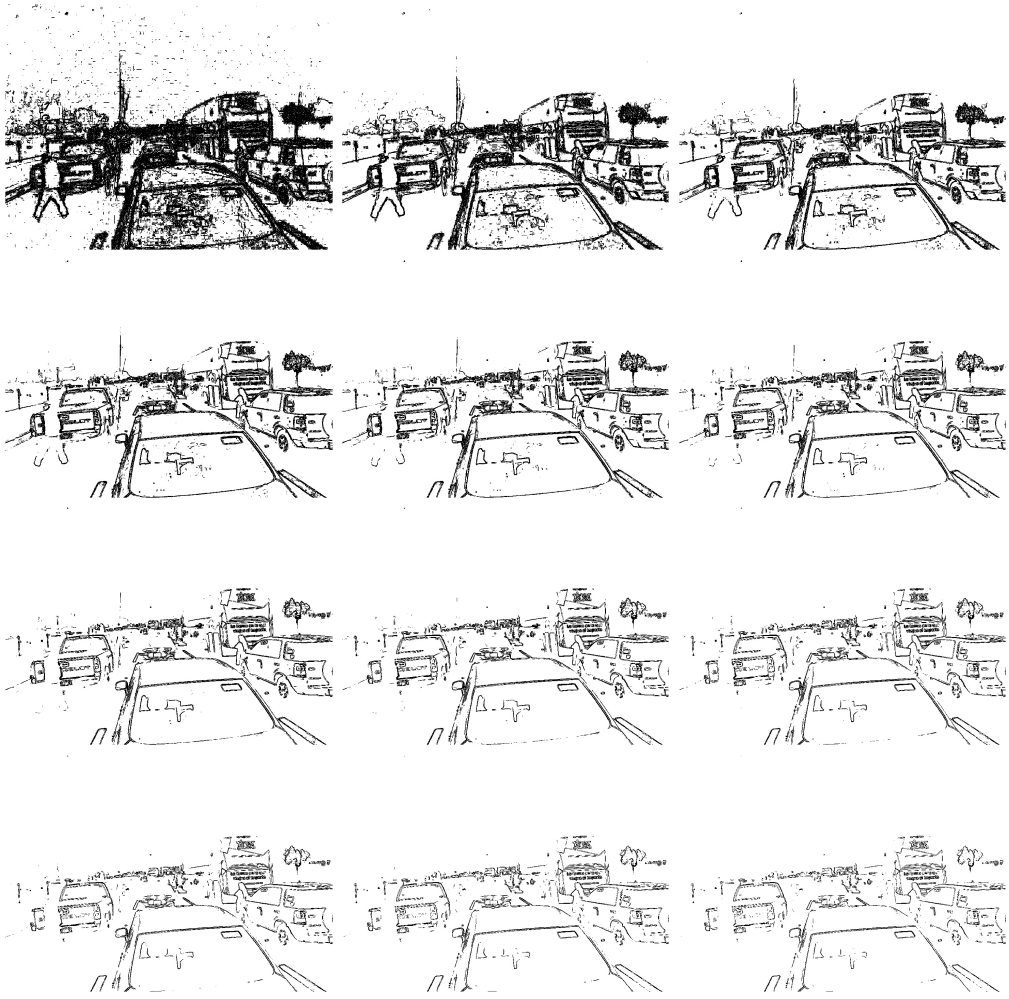

Figura 4.4: Variaciones en el umbral de binarización.

En adelante, la binarización se elegirá según lo requieran las imágenes, podrá apreciarse el umbral elegido en las secciones de código. Vea el tercer parámetro de la función **umbralizar**.

4.2. Diagrama de bloques

Por lo tanto, para procesar una imagen en formato BMP se sigue la siguiente secuencia de funciones;

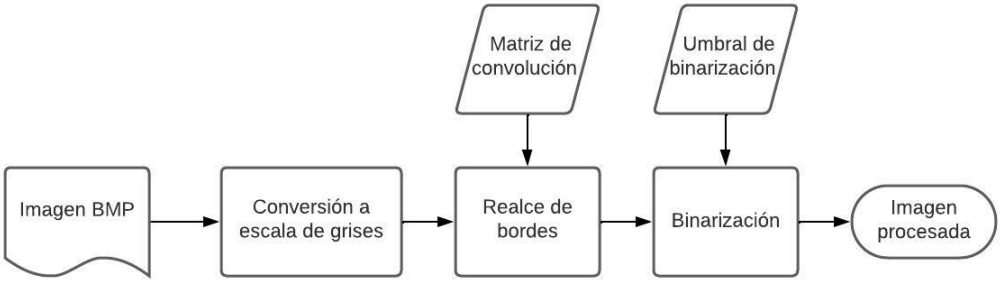

Figura 4.5: Diagrama del procesamiento de una imagen.

Observe que el proceso de binarización puede ocurrir varias veces hasta que se este satisfecho con el umbral. Determinar este umbral puede depender de la mascara de convolución que se use y la propia imagen digital.
Ademas, cada paso del proceso genera una imagen nueva, misma que es utilizada en el siguiente paso, esto es conveniente si se desea tener un registro de las modificaciones que se hacen durante el procesamiento. Si se encuentra preocupado por la cantidad de memoria que esto puede utilizar, puede sobreescribir sobre las imagenes de procesos anteriores.
No se recomienda usar el mismo archivo para la entrada y la salida de una misma función ya que ambos archivos se abren en simultaneo.

Capítulo 5

Resultados

Como parte final de este documento, con el fin de comprender mejor su funcionamiento y para poder comparar los distintos resultados, se presentan algunos ejemplos de la aplicación de las diferentes máscaras de convolución en algunas imágenes.

5.1. Resultados de los operadores de gradiente

En las secciones siguientes se mostraran los resultados que se obtienen con las diferentes máscaras de convolución mediante el algoritmo construido en C, se incluyen las instrucciones utilizadas para mostrar al lector como replicar los resultados con sus propias imágenes si así lo desea.

5.1.1. Operadores de Roberts

Si se regresa a la sección dedicada a los filtros de Roberts se puede recuperar el par de matrices

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

y

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

también note que son matrices de orden 2, de modo que no tienen definido un algoritmo de convolución, sin embargo, si se tiene claro el objetivo de estas matrices, podemos construir un par de matrices de orden 3 que realicen las operaciones que deberían realizarse con las matrices de Roberts. Dependiendo de la dirección en la que se desee aproximar la derivada podemos tener hasta 4 formas de adaptar un filtro de Roberts, la que es quizá la manera más obvia

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

Con ello, se puede proceder a ejecutar

```
1     filtro_gris("img.bmp","Gris.bmp");
2
3     int horizontal[3][3]={{0,0,0},
4                           {0,1,0},
5                           {0,0,-1}};
6
7     convolucionar_abs("Gris.bmp","horizontal.bmp",
         horizontal);
8
9     umbralizar("horizontal.bmp","horizontalUmbra80.bmp
         ",80);
```

para convertir la imagen a escala de grises, pasar la matriz de convolución y posteriormente pasar por un umbral de 80, lo que produce la secuencia de imágenes siguiente

Figura 5.1: Primera secuencia de procesamiento del filtro de Roberts.

Recuerde que la función **umbralizar** ha invertido los colores para que los bordes de la imagen resalten en negro, haciendo más cómodo de ver. Del mismo modose procede con la segunda matriz

```
1     int vertical[3][3]={{0,0,0},
2                         {0,0,1},
3                         {0,-1,0}};
4
5     convolucionar_abs("Gris.bmp","vertical.bmp",vertical
         );
6
7     umbralizar("vertical.bmp","verticalUmbra80.bmp",80);
```

Figura 5.2: Segunda secuencia de procesamiento del filtro de Roberts.

Si se compara los resultados finales del procesamiento, podremos apreciar que el filtro vertical ha tenido algunos problemas para detectar bordes que se encuentran en la misma dirección que el filtro

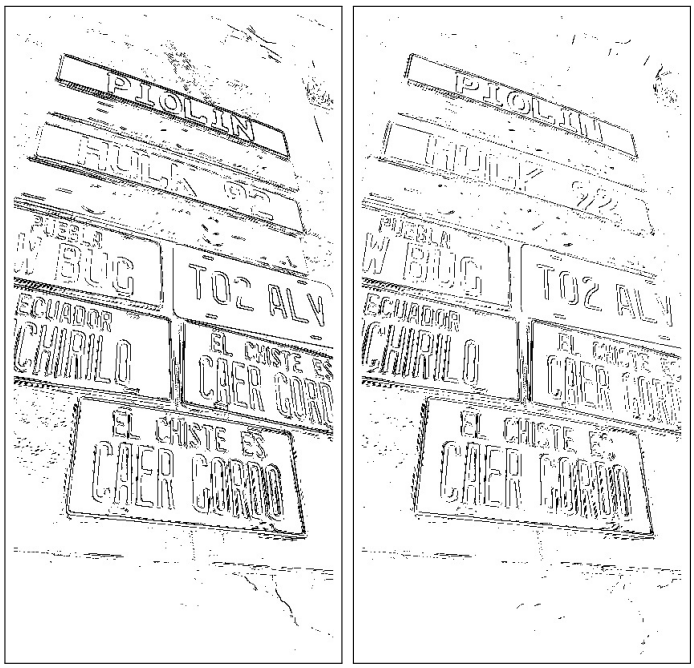

Figura 5.3: Comparación de los filtros de Roberts.

Se puede obtener un resultado de mejor calidad si se hace uso de la función **convolucionar_Magnitud**, esta tomara las aproximaciones de ambas derivadas para calcular la magnitud del vector gradiente, lo que permitirá obtener una magnitud de diferenciación que tenga en cuenta ambas direcciones a la vez

```
1    convolucionar_Magnitud("Gris.bmp","RobersDoble.bmp",
        vertical,horizontal);
2    umbralizar("RobersDoble.bmp","DobleUmbra80.bmp",80);
```

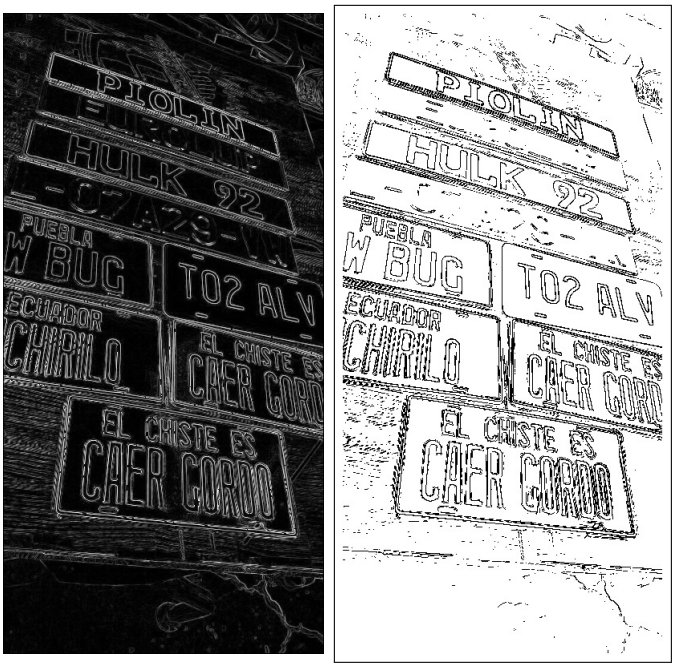

Figura 5.4: Ejecución en conjunto de ambos filtros de Roberts.

El resultado es visiblemente mejor aunque ciertamente, calcular dos convoluciones, añadido al cálculo de una raíz cuadrada puede aumentar considerablemente el costo computacional de procesar la imagen. Para los siguientes operadores, aunque se mantendrán todos los pasos en el código, únicamente se mostraran los resultados posterior a umbralizar, pues resultan más cómodos de ver y el resto de pasos son obvios.

5.1.2. Operadores de Prewitt

Los operadores de Prewitt son, afortunadamente, más simples de implementar pues no requieren tener que hacer adaptaciones de ningún tipo a las matrices, que son

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

y

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

por lo que se puede directamente codificar;

```
1
2
3     int horizontal[3][3]={{-1,0,1},
4                           {-1,0,1},
5                           {-1,0,1}};
6
7     convolucionar_abs("Gris.bmp","horizontal.bmp",
         horizontal);
8
9
10
11    umbralizar("horizontal.bmp","HorizontalUmbra80.bmp
         ",80);
```

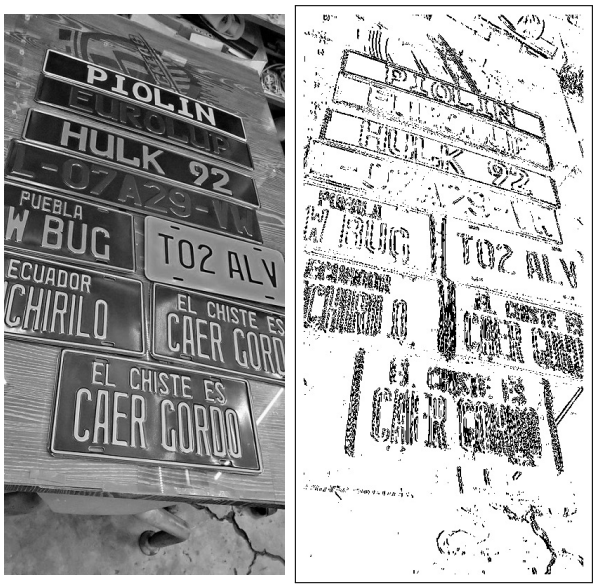

Figura 5.5: Resultados de aplicar el filtro horizontal de Prewitt.

Para el filtro horizontal, que claramente tiene defectos en la detección de bordes en posición horizontal. Así mismo el filtro vertical, presentara problemas para detectar bordes verticales.

```
1     int vertical[3][3]={{-1,-1,-1},
2                         { 0, 0, 0},
3                         { 1, 1, 1}};
4
5
6     convolucionar_abs("Gris.bmp","vertical.bmp",vertical
         );
7
8     umbralizar("vertical.bmp","VerticalUmbra80.bmp",80);
```

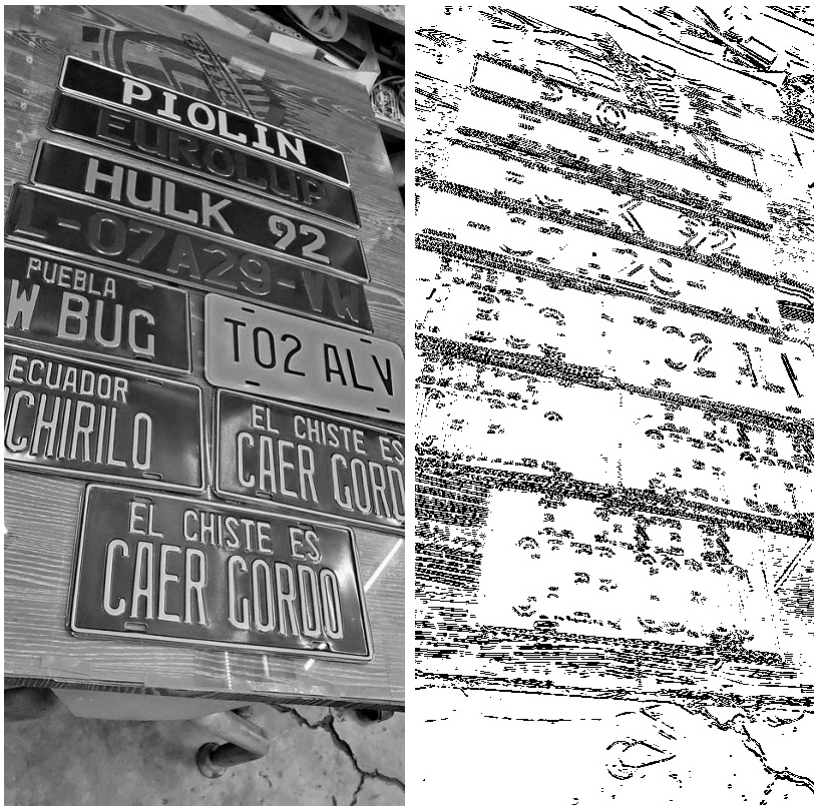

Figura 5.6: Resultados de aplicar el filtro vertical de Prewitt.

Para la magnitud del gradiente la implementación se hace de forma análoga al caso de Roberts, salvo las máscaras.

```
convolucionar_Magnitud("Gris.bmp","Doble.bmp",
   vertical,horizontal);
umbralizar("Doble.bmp","DobleUmbra80.bmp",80);
```

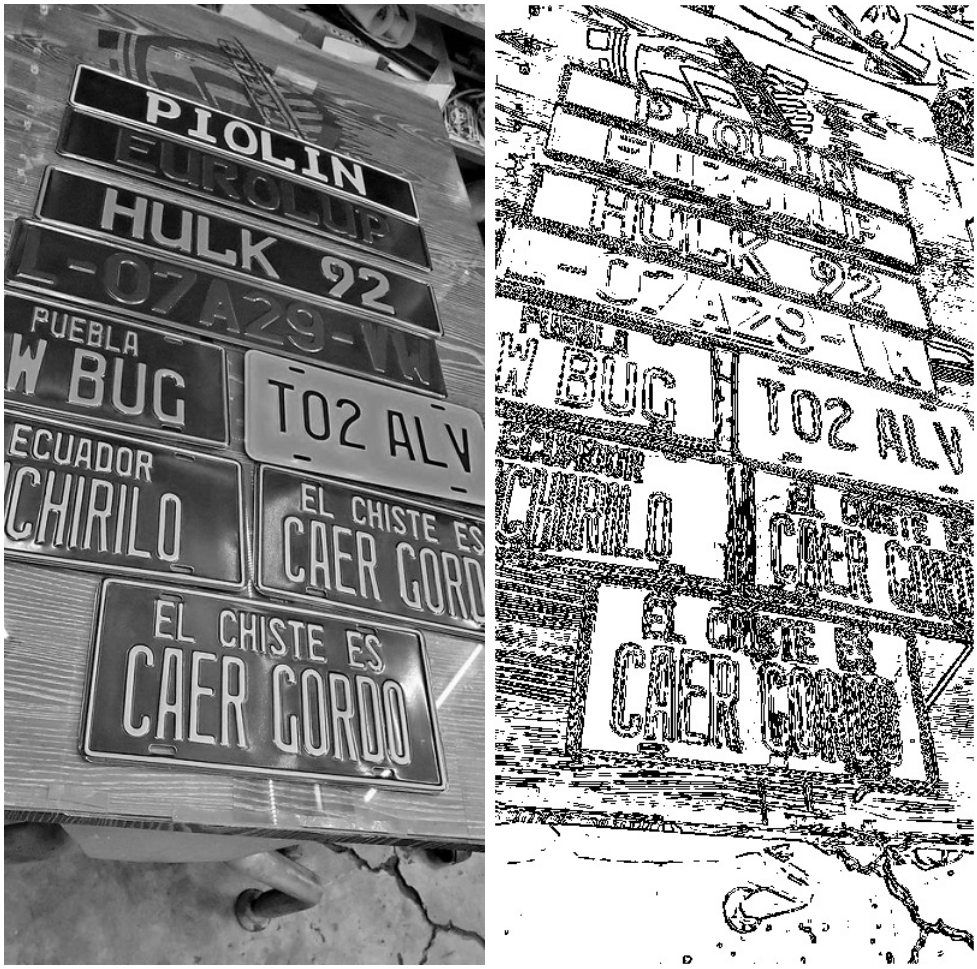

Figura 5.7: Ejecución conjunta del filtro de Prewitt.

Note como nuevamente se logran detectar los bordes tanto verticales como horizontales de manera exitosa.

5.1.3. Operadores de Sobel

Se parte de las matrices que ya se han construido en el capítulo anterior, y se ejecutan los comandos como se ha venido haciendo.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Se ejecuta el script para la primera matriz, que es la diferenciación en horizontal.

```
filtro_gris("pua.bmp","Gris.bmp");




    int horizontal[3][3]={{-1,0,1},
                          {-2,0,2},
                          {-1,0,1}};

    convolucionar_abs("Gris.bmp","horizontal.bmp",
       horizontal);

    umbralizar("horizontal.bmp","HorizontaUmbra80.bmp
       ",80);
```

Figura 5.8: Ejecución del filtro horizontal de Sobel.

Para obtener el resultado con el filtro en vertical se ejecutan las ordenes;

```
1
2     int vertical[3][3]={{-1,-2,-1},
3                         { 0, 0, 0},
4                         { 1, 2, 1}};
5
6     convolucionar_abs("Gris.bmp","vertical.bmp",vertical
         );
7
8     umbralizar("vertical.bmp","VertivcaUmbra80.bmp",80);
```

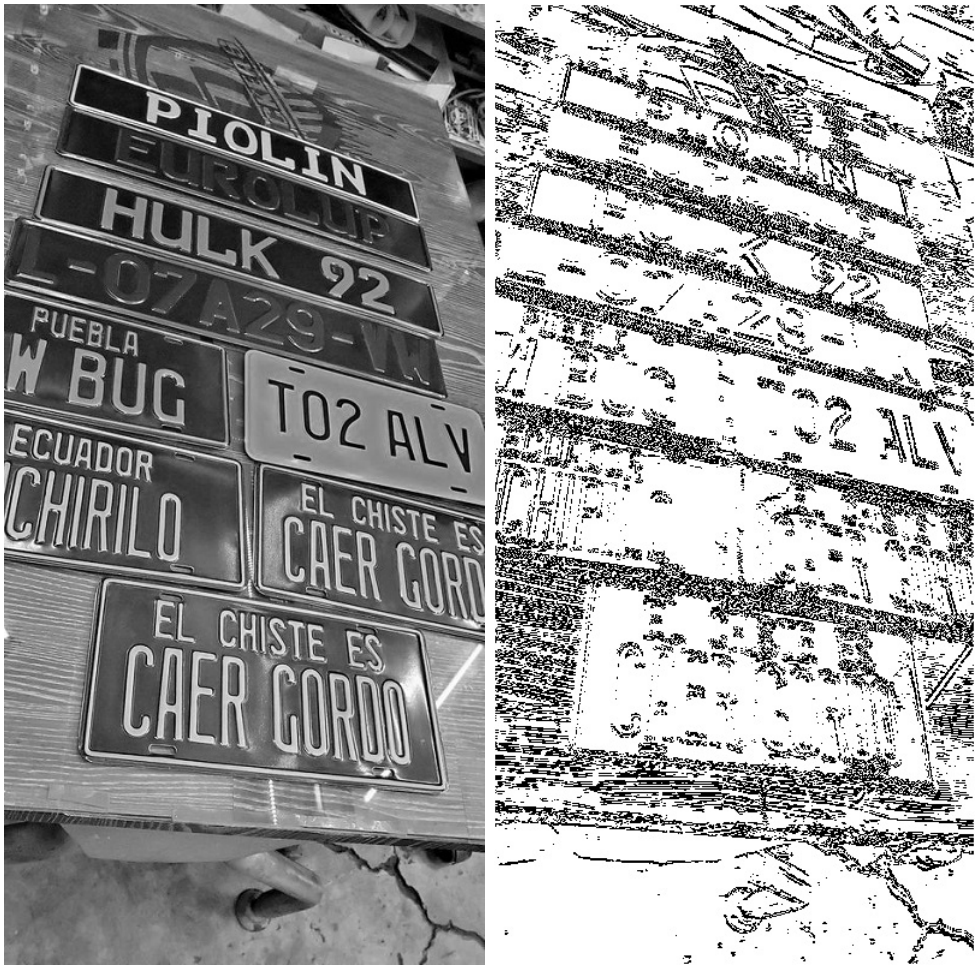

Figura 5.9: Ejecución del filtro vertical de Sobel.

El uso de ambas matrices en simultaneo a través de la magnitud del vector gradiente proporciona un resultado más potente

```
1    convolucionar_Magnitud("Gris.bmp","Doble.bmp",
        vertical,horizontal);
2    umbralizar("Doble.bmp","DobleUmbra80.bmp",80);
```

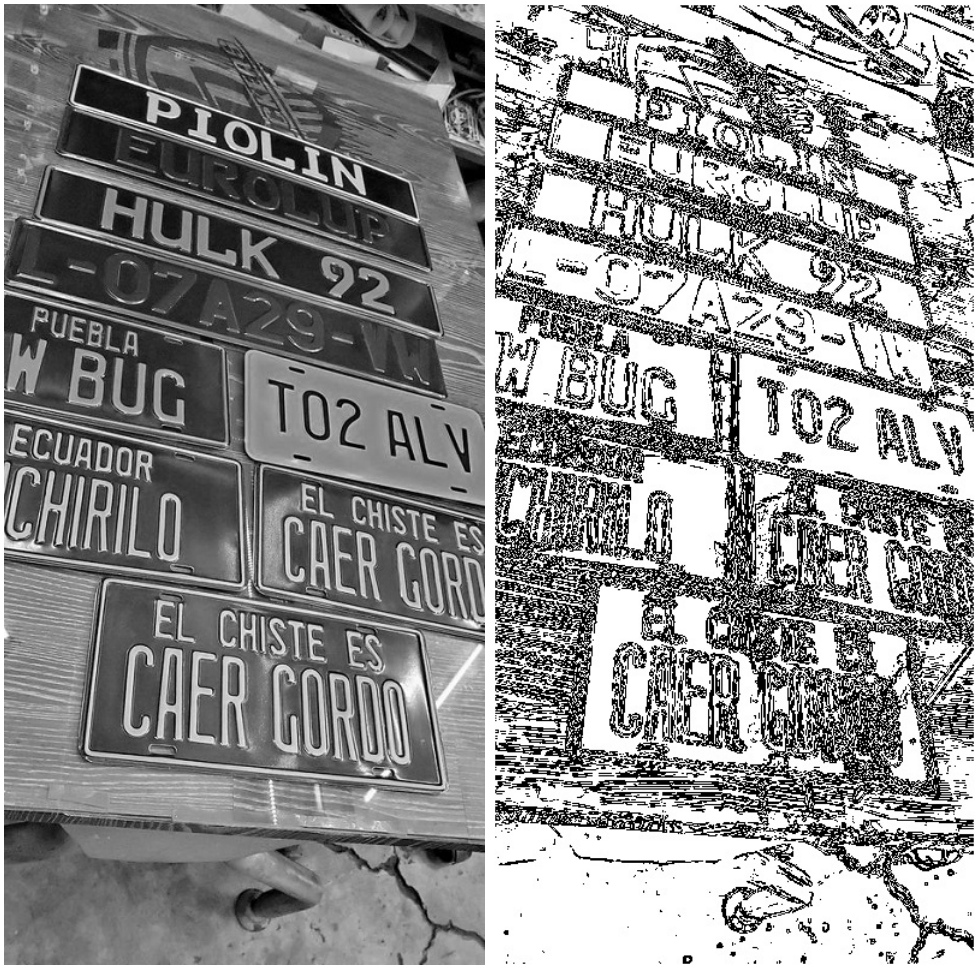

Figura 5.10: Ejecución en simultaneo del filtro de Sobel.

Deacuerdo con los autores, el filtro de Sobel proporciona una mejor respuesta ante el ruido de la imagen al incluir un pequeño efecto de suavizamiento Gaussiano, para poder apreciar el efecto de este suavizamiento, se prueba procesar una imagen con ruido añadido, la imagen en cuestión es la siguiente

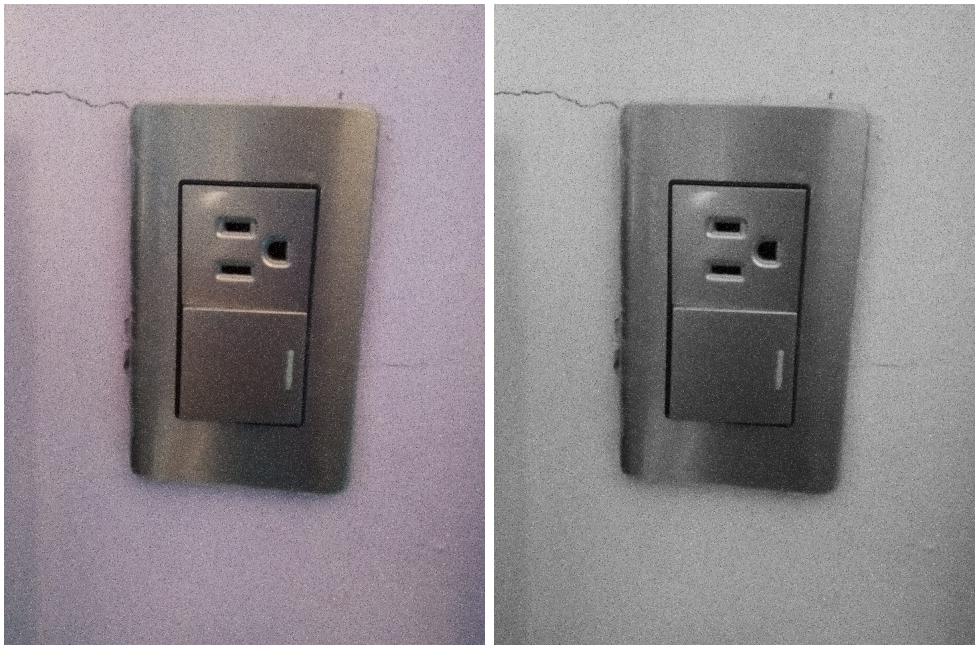

Figura 5.11: Imagen con ruido tipo grano, a la izquierda imagen original, a la derecha imagen en escala de grises.

Si se observa con detenimiento se puede apreciar que hay cierto tipo de grano distribuido por toda la imagen, que claramente no es parte de la imagen real, la aparición de este tipo de defectos por lo general se debe a fallos de cámara o de memoria. Si se pasa esta imagen por un filtro Prewitt se obtiene

```
1     filtro_gris("ruido.bmp","Gris.bmp");
2
3     int horizontal[3][3]={{-1,0,1},
4                           {-1,0,1},
5                           {-1,0,1}};
6
7     int vertical[3][3]={{-1,-1,-1},
8                         { 0, 0, 0},
9                         { 1, 1, 1}};
10
11    convolucionar_Magnitud("Gris.bmp","Prewit.bmp",
         vertical,horizontal);
12
```

```
13      umbralizar("Prewit.bmp","PrewitUmbra30.bmp",30);
```

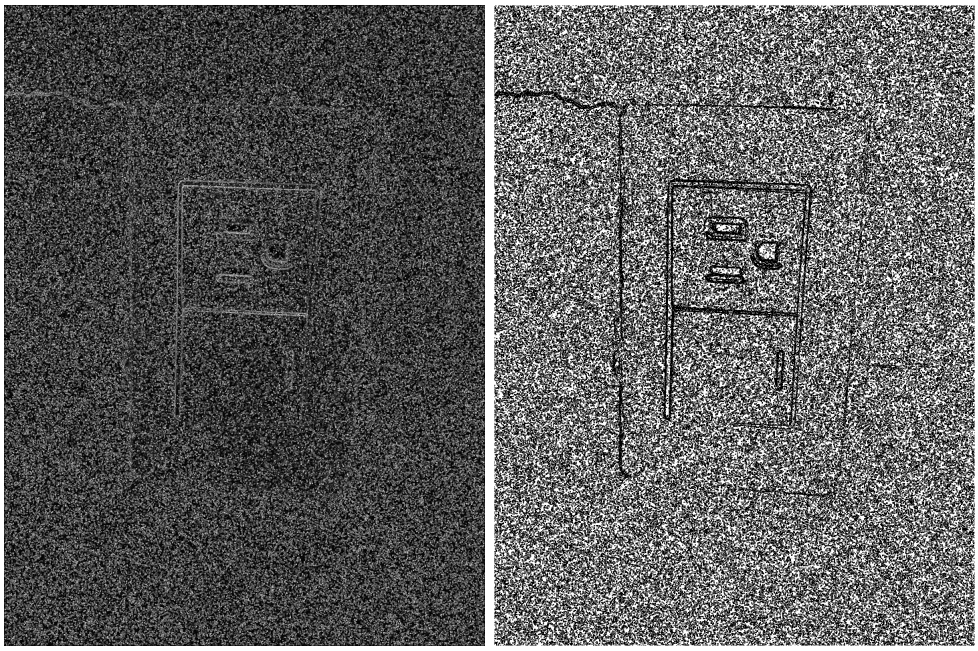

Figura 5.12: Imagen con ruido procesada por el filtro de Prewitt, a la izquierda resultado del filtro, a la derecha misma imagen binarizada.

Por otro lado haciendo uso del filtro de Sobel se tendrá

```
1       filtro_gris("ruido.bmp","Gris.bmp");
2
3       int horizontal[3][3]={{-1,0,1},
4                             {-2,0,2},
5                             {-1,0,1}};
6
7       int vertical[3][3]={{-1,-2,-1},
8                           { 0, 0, 0},
9                           { 1, 2, 1}};
10
11      convolucionar_Magnitud("Gris.bmp","Sobel.bmp",
           vertical,horizontal);
12
```

```
13    umbralizar("Sobel.bmp","SobelUmbra10.bmp",10);
```

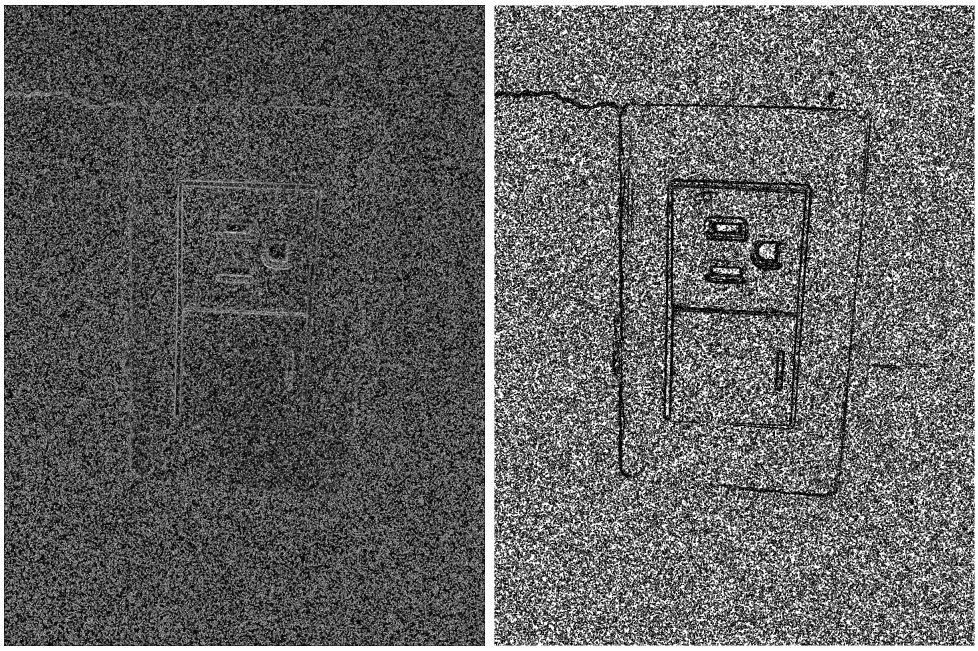

Figura 5.13: Imagen con ruido procesada por el filtro de Sobel, a la izquierda resultado del filtro, a la derecha misma imagen binarizada.

En el siguiente par de imágenes, a la izquierda se encuentra el resultado del filtro de Prewitt y a la derecha el de Sobel, viendo ambos resultados juntos se puede notar que ciertamente, el filtro de Sobel ha tenido resultados visiblemente mejores que el filtro de Prewitt en imágenes con ruido.

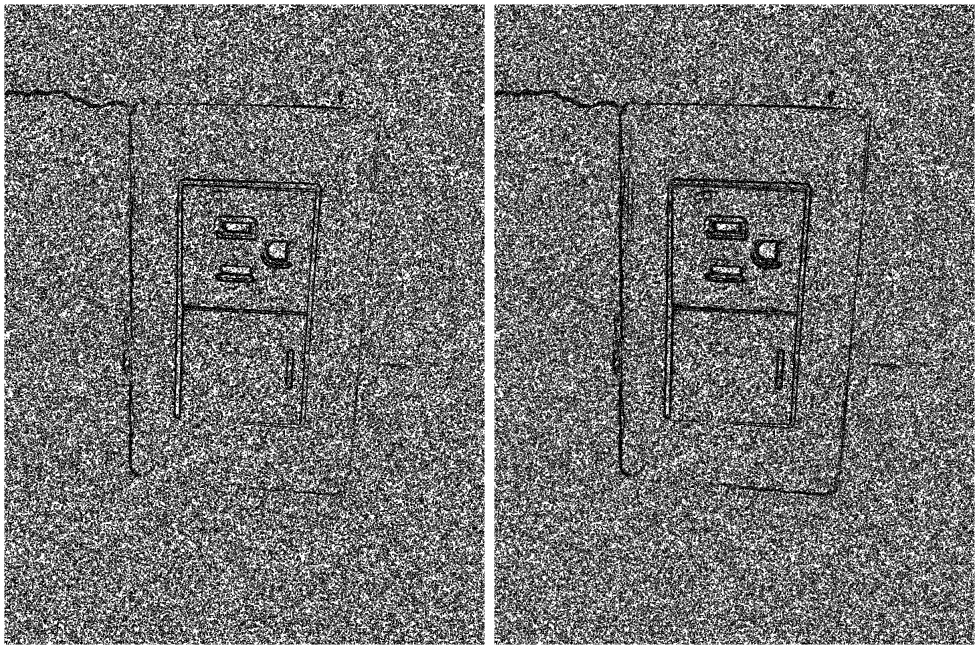

Figura 5.14: Comparación de respuesta al ruido de los filtros Prewitt (Izquierda) contra Sobel (Derecha).

5.1.4. Operador Laplaciano

El operador Laplaciano tiene como característica a destacar que no requiere la aplicación de dos filtros en simultaneo, lo cual es la razón de existir de una función de convolución especializada, es decir, se puede ejecutar el filtro Laplaciano usando la función de convolución estándar, obviamente hay que seguir calculando el valor absoluto de cada operación.

```
1     filtro_gris("control.bmp","Gris.bmp");
2
3     int laplaciano[3][3]={{ 0,-1,0},
4                           { -1,4,-1},
5                           { 0,-1,0}};
6
7     convolucionar_abs("Gris.bmp","laplaciano.bmp",
         laplaciano);
```

```
8
9       umbralizar("laplaciano.bmp","laplaciano6.bmp",6);
```

Figura 5.15: Secuencia de procesado por el filtro Laplaciano.

Como se menciona antes, la principal debilidad de este filtro, en contrapeso con su velocidad de implementación es la duplicación de las orillas dada la naturaleza de la segunda derivada, veamos una comparación con el filtro de Prewitt, que no añade ningún tipo de suavizamiento. Note que el umbral en el filtro de Prewitt es 6 veces mayor que en el del filtro Laplaciano considerando que la primera calcula 3 derivadas y omite la división por 2 en cada una, mientras que la segunda es exacta.

```
1       filtro_gris("control.bmp","Gris.bmp");
2
3       int horizontal[3][3]={{-1,0,1},
4                             {-1,0,1},
5                             {-1,0,1}};
6
7       int vertical[3][3]={{-1,-1,-1},
8                           { 0, 0, 0},
9                           { 1, 1, 1}};
10
11      convolucionar_Magnitud("Gris.bmp","Prewit.bmp",
           vertical,horizontal);
12
13      umbralizar("Prewit.bmp","PrewitUmbra12.bmp",36);
14
15      int laplaciano[3][3]={{ 0,-1,0},
```

```
16                              {  -1,4,-1},
17                              {  0,-1,0}};
18
19      convolucionar_abs("Gris.bmp","laplaciano.bmp",
           laplaciano);
20
21      umbralizar("laplaciano.bmp","laplaciano6.bmp",6);
```

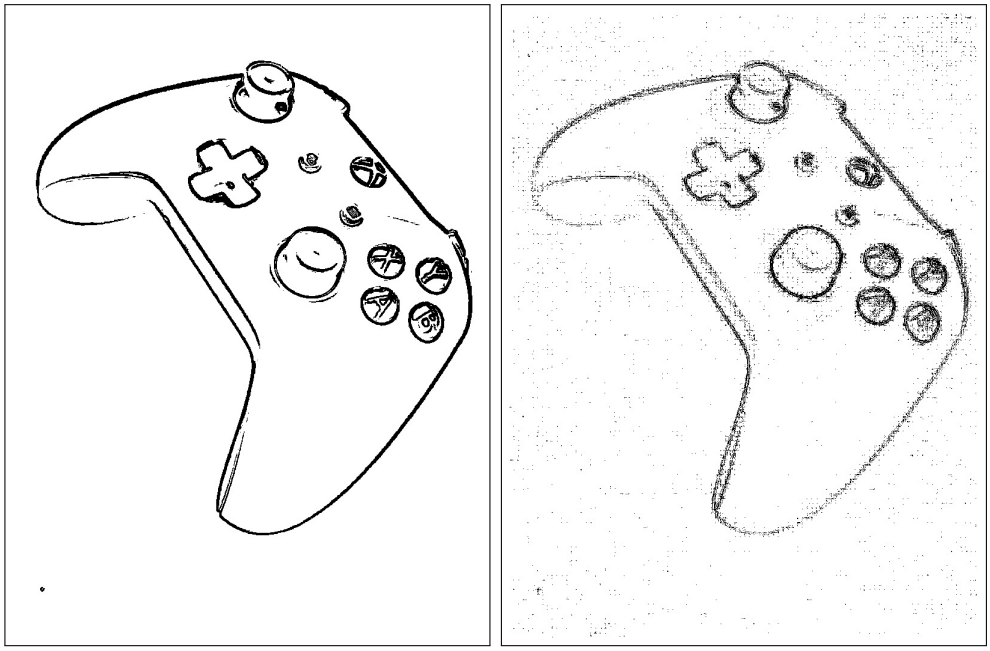

Figura 5.16: Comparación de imágenes procesadas por el filtro de Prewitt (Izquierda) contra el filtro Laplaciano (Derecha).

Se puede ver entonces que, aunque menos costoso, el filtro Laplaciano presenta algunos inconvenientes, (Sucar y Gómez, 2011) mencionan la existencia de un algoritmo de eliminación de orillas dobles, pero esto implicaría sumar costo computacional, a mi parecer despojándolo de la única ventaja que posee.

5.1.5. Operadores de Kirsch

Evaluar los operadores de Kirsch es un proceso particular, pues el algoritmo consiste en evaluar las cuatro matrices direccionales de 0, 45, 90 y 135 grados y tomar el valor más alto de las 4 convoluciones, como no se dispone de ninguna función que admita cuatro matrices a la vez y dado que este algoritmo es el único que lo requiere, se ha construido la función

```
int filtroKirch(char nombre[], char out[]);
```

Que directamente se encarga de aplicar el filtro de Kirsch. La manera de hacer uso es de esta es análogo a los otros algoritmos, un ejemplo de este es como sigue

```
1 filtro_gris("control.bmp","Gris.bmp");
2
3 filtroKirch("Gris.bmp","Kirch.bmp");
4
5 umbralizar("Kirch.bmp","Kirch36.bmp",36);
```

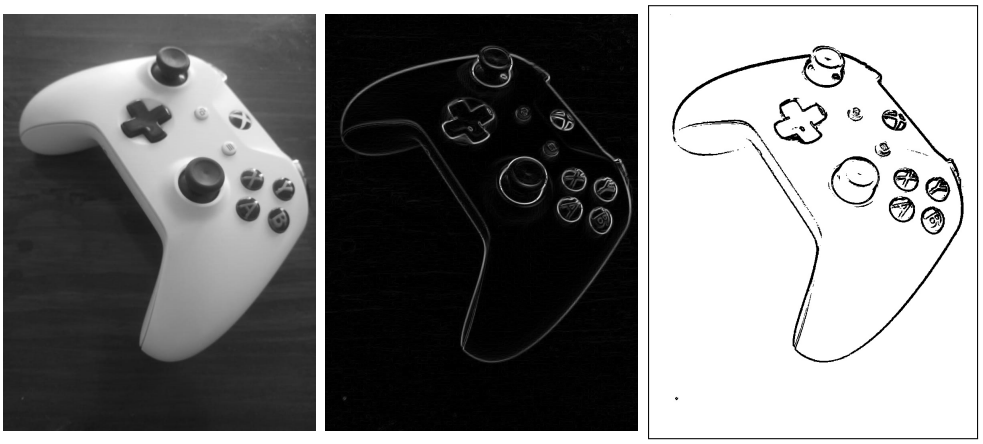

Figura 5.17: Secuencia de procesamiento del filtro de Kirsch.

Si se desea observar por separado cada uno de los filtros direccionales bastara hacer uso de la función de convolución estándar **convolucinoar_abs** con la respectiva matriz

```
    filtro_gris("control.bmp","Gris.bmp");


    int horizontal[3][3]={{-1,0,1},
                          {-1,0,1},
                          {-1,0,1}};

    int vertical[3][3]={{-1,-1,-1},
                        { 0, 0, 0},
                        { 1, 1, 1}};

    int kirch45[3][3]=  {{ 0, 1, 1},
                        {-1, 0, 1},
                        {-1,-1, 0}};

    int kirch135[3][3]=  {{ -1,-1, 0},
                        { -1, 0, 1},
                        {  0, 1, 1}};

    convolucionar_abs("Gris.bmp","vertical.bmp",vertical
       );
    umbralizar("vertical.bmp","vertical36.bmp",36);

    convolucionar_abs("Gris.bmp","horizontal.bmp",
       horizontal);
    umbralizar("horizontal.bmp","horizontal36.bmp",36);

    convolucionar_abs("Gris.bmp","45grados.bmp",kirch45)
       ;
    umbralizar("45grados.bmp","45grados36.bmp",36);

    convolucionar_abs("Gris.bmp","135grados.bmp",
       kirch135);
    umbralizar("45grados.bmp","135grados36.bmp",36);
```

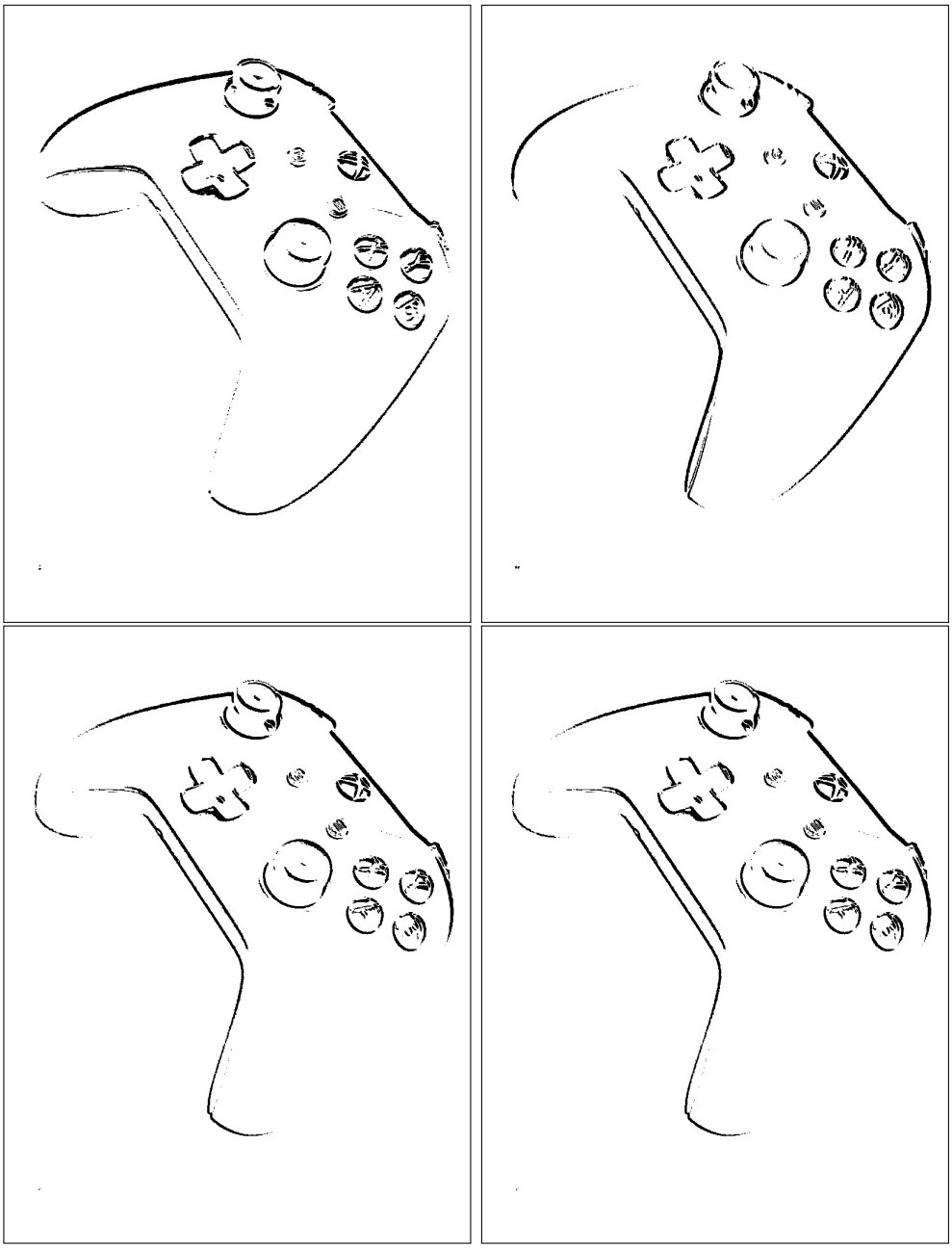

Figura 5.18: Ejecución de las matrices discrecionales de Kirsch por separado de izquierda a derecha y de arriba a abajo; vertical, horizontal, 45° y 135°.

5.2. Otros resultados

Esta sección esta dedicada a probar las matrices de convolución que construyeron como pasos previos a las matrices que ostentan un nombre propio, así como explorar que ocurre cuando las matrices admiten coeficientes que no necesariamente son enteros. Como primer paso se recuerdan las matrices de diferencias progresivas y regresivas previas a las diferencias cruzadas que usa Roberts.

$$\begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$$

Así mismo las diferencias regresivas

$$\begin{bmatrix} 0 & 0 \\ -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}$$

Al igual que con Roberts, se adaptan estas matrices a matrices de convolución de orden 3 para obtener las diferencias progresivas

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Se ejecuta el algoritmo para las matrices por separado

```
1     filtro_gris("control.bmp","Gris.bmp");
2
3
4     int horizontal[3][3]={{0, 0,0},
5                           {0,-1,1},
6                           {0, 0,0}};
```

```
7
8       int vertical[3][3]={{ 0, 0, 0},
9                           { 0,-1, 0},
10                          { 0, 1, 0}};
11
12      convolucionar_abs("Gris.bmp","horizontal.bmp",
           horizontal);
13      umbralizar("horizontal.bmp","horizontal36.bmp",12);
14
15      convolucionar_abs("Gris.bmp","vertical.bmp",vertical
           );
16      umbralizar("vertical.bmp","vertical36.bmp",12);
```

Figura 5.19: Ejecución de procesado por diferencias hacia adelante, a la izquierda el filtrado horizontal, a la derecha el vertical.

y para las matrices en conjunto

```
1       convolucionar_Magnitud("Gris.bmp","doble.bmp",
           vertical,horizontal);
```

```
2     umbralizar("doble.bmp","doble24.bmp",12);
```

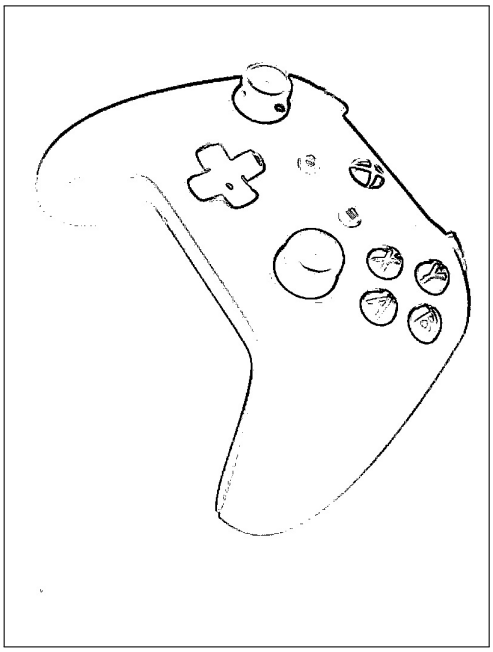

Figura 5.20: Procesado dual de las diferencias hacia enfrente.

Se hace lo mismo para las diferencias regresivas

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
1     filtro_gris("control.bmp","Gris.bmp");
2
3
4     int horizontal[3][3]={{0, 0,0},
5                           {-1,1,0},
6                           {0, 0,0}};
```

```
7
8       int vertical[3][3]={{ 0,-1, 0},
9                           { 0, 1, 0},
10                          { 0, 0, 0}};
11
12      convolucionar_abs("Gris.bmp","horizontal.bmp",
           horizontal);
13      umbralizar("horizontal.bmp","horizontal36.bmp",12);
14
15
16      convolucionar_abs("Gris.bmp","vertical.bmp",vertical
           );
17      umbralizar("vertical.bmp","vertical36.bmp",12);
18
19
20      convolucionar_Magnitud("Gris.bmp","doble.bmp",
           vertical,horizontal);
21      umbralizar("doble.bmp","doble24.bmp",12);
```

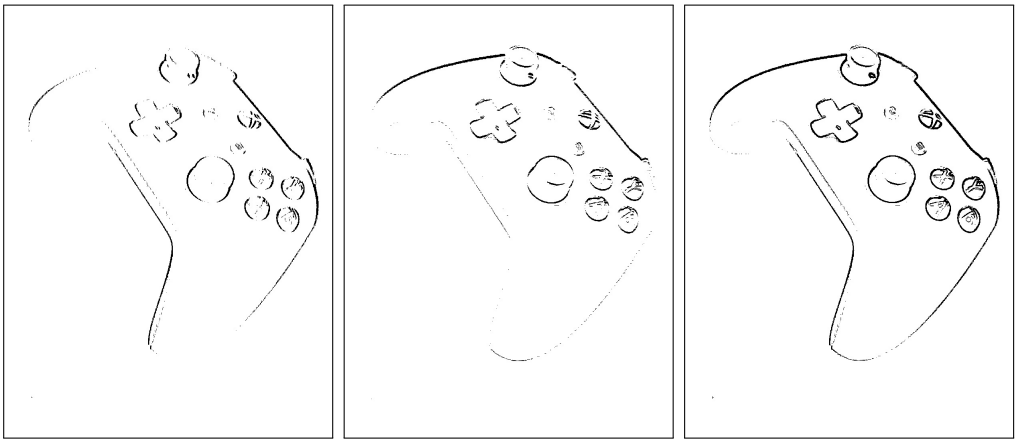

Figura 5.21: Filtrado por diferencias hacia regresivas, de izquierda a derecha, filtrado horizontal, filtrado vertical, filtrado dual.

Por otro lado, antes de presentar las máscaras de Prewitt, se mencionan un par de versiones simplificadas

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

```
    filtro_gris("control.bmp","Gris.bmp");


    int horizontal[3][3]={{0, 0,0},
                          {-1,0,1},
                          {0, 0,0}};

    int vertical[3][3]={{ 0,-1, 0},
                        { 0, 0, 0},
                        { 0, 1, 0}};

    convolucionar_abs("Gris.bmp","horizontal.bmp",
       horizontal);
    umbralizar("horizontal.bmp","horizontal12.bmp",12);


    convolucionar_abs("Gris.bmp","vertical.bmp",vertical
       );
    umbralizar("vertical.bmp","vertical12.bmp",12);


    convolucionar_Magnitud("Gris.bmp","doble.bmp",
       vertical,horizontal);
    umbralizar("doble.bmp","doble24.bmp",12);
```

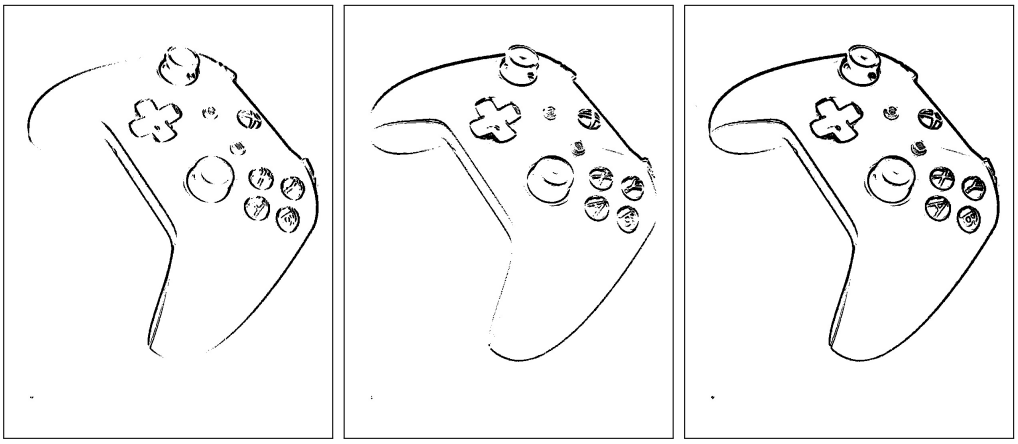

Figura 5.22: Ejecución del filtro simplificado de Prewitt, de izquierda a derecha, horizontal, vertical y dual.

Aunque el filtro Gaussiano solo es mencionado brevemente como paso previo al filtro de Sobel, para no dejar espacio a la curiosidad, se presenta un ejemplo de los efectos del mismo sobre una imagen con ruido. Note el uso de la función **convolucionar** que es la convolución pura.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 8 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

```
1     filtro_gris("ruido.bmp","Gris.bmp");
2
3     int gauss[3][3]=      {{1,2,1},
4                            {2,8,2},
5                            {1,2,1}};
6
7     convolucionar("Gris.bmp","Gaussian.bmp",gauss);
```

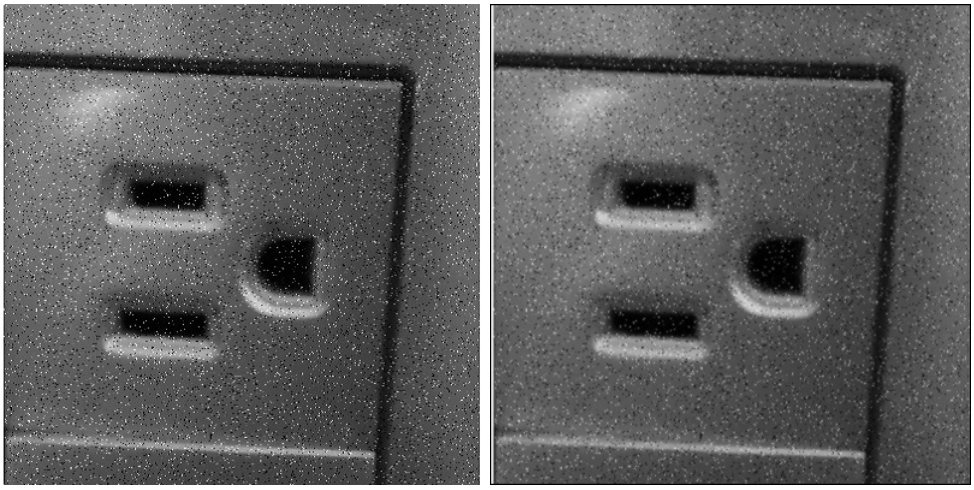

Figura 5.23: Ejecución de filtro Gaussiano, a la izquierda imagen con ruido, a la derecha imagen filtrada.

El filtro Laplaciano resulta de la suma de dos matrices, que calculan, respectivamente la segunda derivada parcial en x y la segunda derivada parcial en y

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

```
1     filtro_gris("control.bmp","Gris.bmp");
2
3
4     int horizontal[3][3]={{0, 0,0},
5                           {1,-2,1},
6                           {0, 0,0}};
7
8     int vertical[3][3]={{ 0, 1, 0},
9                         { 0,-2, 0},
10                        { 0, 1, 0}};
```

```
11
12      convolucionar_abs("Gris.bmp","horizontal.bmp",
           horizontal);
13      umbralizar("horizontal.bmp","horizontal5.bmp",5);
14
15
16      convolucionar_abs("Gris.bmp","vertical.bmp",vertical
           );
17      umbralizar("vertical.bmp","vertical5.bmp",5);
18
19
20      convolucionar_Magnitud("Gris.bmp","doble.bmp",
           vertical,horizontal);
21      umbralizar("doble.bmp","doble5.bmp",5);
```

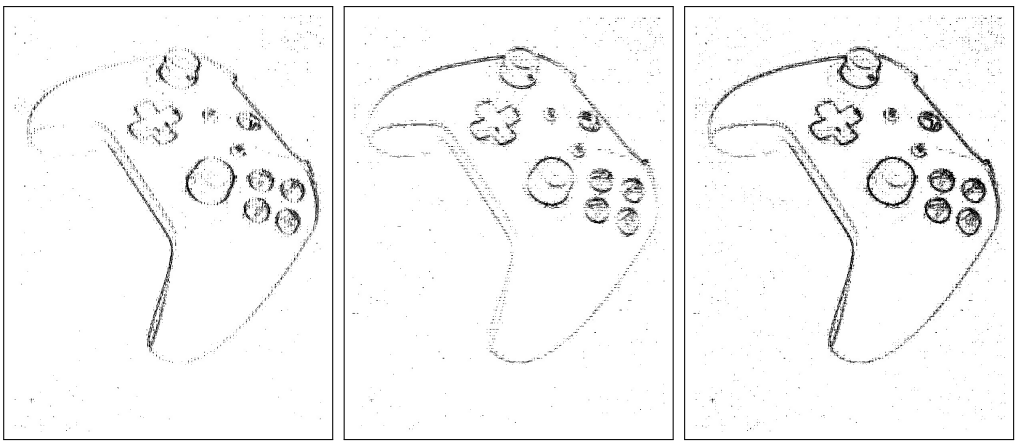

Figura 5.24: Filtrado por medio de segunda derivada en horizontal, vertical y considerando ambas direcciones.

5.2.1. Operadores con coeficientes fraccionarios

Existen gran variedad de máscaras de convolución que hacen uso de coeficientes fraccionarios, sin embargo, hasta ahora las funciones de convolución reciben como argumentos matrices de enteros, esto por que todas las matrices que hemos estudiado son así, sin embargo se hace necesario implementar una función que permita esto por dos razones, la primera es dar libertad al usua-

rio de explorar la convolución hacia otras direcciones y la segunda evaluar una matriz en particular.

```
1 int convolucionar_float(char nombre[],char out[], float
     matrix[][3]);
```

La función tiene una funcionalidad idéntica a la función **convolucionar_abs** salvo que el argumento matricial admite valores de tipo flotante, esto sin embargo no significa que la imagen resultante tendrá niveles de gris flotantes. Es sabido ya que los datos de un BMP deben ser enteros, el lenguaje C, de manera natural trunca los datos flotantes cuando se pasan a una variable de tipo entero, este efecto es comparable a la conocida función suelo.

Aun con estas limitaciones, nos permitirá hacer uso de la matriz

$$\begin{bmatrix} 0 & 0 & 0 \\ -0{,}5 & 0 & 0{,}5 \\ 0 & 0 & 0 \end{bmatrix}$$

Que tiene la curiosa cualidad de que calcula

$$\frac{\partial f(x_i, y_j)}{\partial x} = \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{2\Delta x} = \frac{f(x_{i+1}, y_j) - f(x_{i-1}, y_j)}{2}$$

de manera exacta, con todo y la división, algo que se había tenido que tener en cuenta en el procesamiento posterior de binarizacion. Los resultados no son distintos a sus versiones enteras, pero proveen la satisfacción de hacer un cálculo más exacto.

```
1     filtro_gris("control.bmp","Gris.bmp");
2     float flotante[3][3]=     {{   0, 0,  0},
3                                {-0.5, 0,0.5},
4                                {   0, 0,  0}};
5
6     convolucionar_float("Gris.bmp","flotante.bmp",
        flotante);
7
8     umbralizar("flotante.bmp","flotante10.bmp",10);
```

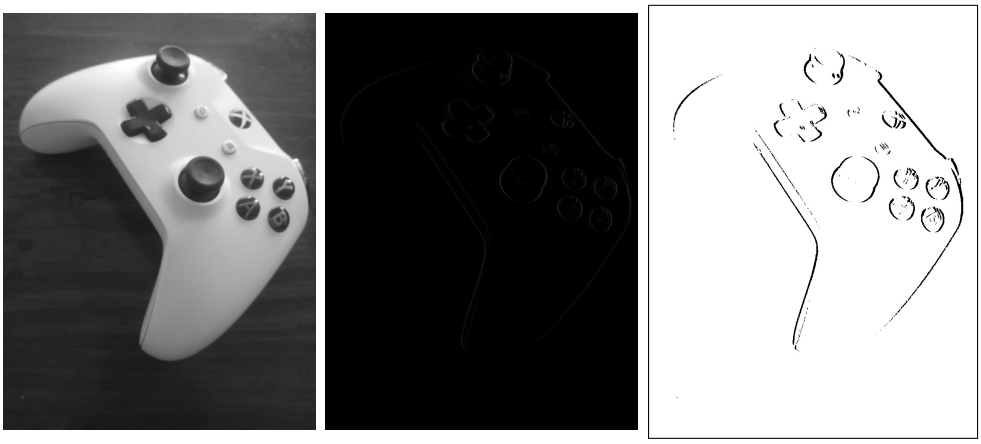

Figura 5.25: Secuencia de procesado por filtro de convolución fraccionario.

Conclusiones

Las matrices de Roberts son incomodas de usar, en cambio Prewitt puede aplicarse directamente sin adaptaciones extrañas, aun así, es verdad que presentan problemas a la hora de enfrentarse al ruido. Los dispositivos fotográficos no son siempre perfectos, y eventualmente uno se vera en la necesidad de procesar imágenes con ruido, lo que lleva a la casi absoluta preferencia por el filtro de Sobel.

El operador Laplaciano es sumamente interesante pero provee resultados poco convincentes, a menos que se use un procesado que remueva orillas dobles, lo cual puede resultar laborioso. Kirsch también provee de resultados convincentes y utiliza un método cuanto menos extravagante.

Considero el trabajo de fundamentación de la teoría que sustenta estos métodos como exitoso, habiendo eliminado hasta donde he sido capaz, las partes poco claras de la construcción de estas matrices, sin embargo, solo el tiempo, y los lectores con poca experiencia en la parte rigurosa de las matemáticas lo dirán.

Referencias

Referencias

Angoa, J., Contreras, A., Ibarra, M., Linares, R., y Martínez, A. (2008). Matemáticas elementales. *México: Fomento Editorial BUAP*.

Cruz Contreras, A., González Robles, J. C., y Herrera Lozada, J. C. (2004). Procesamiento de imágenes: Estructura de archivos bmp. *Polibits*, *30*, 6–8.

Ledesma, A. C., Rosas, K. I. G., y Bernal, O. M. (2015). Introducción al método de diferencias finitas y su implementación computacional. *Bernal, Facultad de Ciencias, UNAM*.

Marsden, J. E., Tromba, A. J., y Mateos, M. L. (1991). *Cálculo vectorial* (Vol. 69). Addison-Wesley Iberoamericana México.

Morales, D. P. (s.f.). Técnicas de filtrado por mascara de convolucion y segmentación de color para procesamiento digital de imágenes.

Oltra, G. N., y Mellado, J. R. A. (2008). *Una introducción a la imagen digital y su tratamiento*. Univ de Castilla La Mancha.

Spivak, M. (1988). *Cálculo infinitesimal*. Reverté.

Sucar, L. E., y Gómez, G. (2011). Visión computacional. *Instituto Nacional de Astrofísica, Optica y Electrónica. Puebla, México*.

Apéndice

5.3. Código para la función de convolución

```
int convolucionar(char nombre[],char out[],int matrix
   [][3]){
    //Inversion vertical de la matriz
    int aux1=matrix[0][0], aux2=matrix[0][1], aux3=
       matrix[0][2];
    matrix[0][0]=matrix[2][0];
    matrix[0][1]=matrix[2][1];
    matrix[0][2]=matrix[2][2];
    matrix[2][0]=aux1;
    matrix[2][1]=aux2;
    matrix[2][2]=aux3;

    //Apertura de archivos
    FILE *img1, *img2;
    int cabecera[54];
    img1=fopen(nombre,"r");
        img2=fopen(out,"w");
    //Volcado de cabecera
    for(int i= 0;i<54;i++){
        cabecera[i]=fgetc(img1);
        fputc(cabecera[i],img2);
    }
    //Obtencion de dimensiones de la imagen
    int ancho = cabecera[18] +cabecera[19] * 256  +
                cabecera[20] * 256  * 256 +cabecera[21]
                   * 256 * 256 * 256;
    int alto = cabecera[22] +cabecera[23] * 256  +
```

```
25                cabecera[24] * 256  * 256 +cabecera[25] *
                      256 * 256 * 256;
26
27 //Reservado de memoria para imagenes
28     fflush(stdin);
29     int i,j;
30
31     int **imagen;
32     imagen = (int **)malloc(alto*sizeof(int*));
33     for (i=0;i<alto;i++) {
34         imagen[i] = (int*)malloc(ancho*sizeof(int));
35     }
36
37     //Volcado de imagen a matriz en memoria
38     int a,b,c;
39     for(i=0;i<alto;i++){
40         for(j=0;j<ancho;j++){
41             a=fgetc(img1);
42             b=fgetc(img1);
43             c=fgetc(img1);
44
45             if(a==b && b==c){
46                   imagen[i][j]=a;
47             }else{
48                printf("\nERROR %d = %d = %d",a,b,c);
49             }
50         }
51     }
52
53     //Reservado de memoria para matriz resultado
54     int **imagen2;
55     imagen2 = (int **)malloc(alto*sizeof(int*));
56     for (i=0;i<alto;i++) {
57         imagen2[i] = (int*)malloc(ancho*sizeof(int));
58     }
59     //Calcular C
60     c=0;
61     for(i=0;i<3;i++){
62         for(j=0;j<3;j++){
63             c=c+matrix[i][j];
```

```
64          }
65      }
66
67      if(c==0){c=1;}
68
69      //Convolucion
70      for(i=1;i<alto-1;i++){
71          for(j=1;j<ancho-1;j++){
72              imagen2[i][j]=imagen[i][j]*matrix[1][1]+
                    imagen[i-1][j]*matrix[0][1]+
73              imagen[i][j-1]*matrix[1][0]+imagen[i-1][j
                    -1]*matrix[0][0]+
74              imagen[i+1][j]*matrix[2][1]+imagen[i][j+1]*
                    matrix[1][2]+
75              imagen[i+1][j+1]*matrix[2][2]+imagen[i-1][j
                    +1]*matrix[0][2]+
76              imagen[i+1][j-1]*matrix[2][0];
77                  imagen2[i][j]=imagen2[i][j]/c;
78          }
79      }
80      //Volcado de matriz procesada a segundo archivo
81      for(i=0;i<alto;i++){
82          for(j=0;j<ancho;j++){
83              fputc(imagen2[i][j],img2);
84              fputc(imagen2[i][j],img2);
85              fputc(imagen2[i][j],img2);
86          }
87      }
88      fclose(img1);
89      fclose(img2);
90      return 0;
91  }
```

5.4. Código para la función para filtro a escala de grises

```
1  int filtro_gris(char nombre[], char out[]){
2      FILE *img1, *img2;
```

```
3       int cabecera[54];
4
5       img1=fopen(nombre,"r");
6           img2=fopen(out,"w");
7
8       for(int i= 0;i<54;i++){
9           cabecera[i]=fgetc(img1);
10          fputc(cabecera[i],img2);
11      }
12
13      int r,g,b;
14      do{ g=fgetc(img1);
15          b=fgetc(img1);
16          r=fgetc(img1);
17
18          fputc((g+b+r)/3,img2);
19          fputc((g+b+r)/3,img2);
20          fputc((g+b+r)/3,img2);
21      }while((r!=EOF && g!=EOF && b!=EOF));
22
23      fclose(img1);
24      fclose(img2);
25      return 0;
26  }
27  \end{verbatim}
28
29  \section{C digo para la funci n de binarizaci n}
30  \begin{lstlisting}
31  int umbralizar(char nombre[],char out[], int umbral){
32      FILE *img1, *img2;
33      int cabecera[54];
34      img1=fopen(nombre,"r");
35          img2=fopen(out,"w");
36
37      for(int i= 0;i<54;i++){
38          cabecera[i]=fgetc(img1);
39          fputc(cabecera[i],img2);
40      }
41      int r,g,b;
42      do{
```

```
43          g=fgetc(img1);
44          b=fgetc(img1);
45          r=fgetc(img1);
46
47          if((g+g+b)/3<umbral){
48              fputc((255),img2);
49              fputc((255),img2);
50              fputc((255),img2);
51          }else{
52              fputc((0),img2);
53              fputc((0),img2);
54              fputc((0),img2);
55          }
56      }while((r!=EOF && g!=EOF && b!=EOF));
57
58      fclose(img1);
59      fclose(img2);
60      return 0;
61  }
```

5.5. Código para el filtro de Kirch

```
1  int filtroKirch(char nombre[], char out[]){
2      int vertical[3][3]={{ 1, 1, 1},
3                          { 0, 0, 0},
4                          {-1,-1,-1}};
5
6      int horizontal[3][3]={{-1,0,1},
7                            {-1,0,1},
8                            {-1,0,1}};
9
10     int kirch1[3][3]=  {{ 0, 1, 1},
11                         {-1, 0, 1},
12                         {-1,-1, 0}};
13
14     int kirch2[3][3]=  {{ -1,-1, 0},
15                         { -1, 0, 1},
16                         {  0, 1, 1}};
```

```
17      FILE *img1, *img2;
18      int cabecera[54];
19      img1=fopen(nombre,"r");
20      img2=fopen(out,"w");
21
22      for(int i= 0;i<54;i++){
23          cabecera[i]=fgetc(img1);
24          fputc(cabecera[i],img2);
25      }
26      int ancho = cabecera[18] +cabecera[19] * 256  +
27                  cabecera[20] * 256  * 256 +cabecera[21]
                       * 256 * 256 * 256;
28      int alto = cabecera[22] +cabecera[23] * 256  +
29                 cabecera[24] * 256  * 256 +cabecera[25] *
                       256 * 256 * 256;
30
31      fflush(stdin);
32      int i,j;
33
34      int **imagen;
35      imagen = (int **)malloc(alto*sizeof(int*));
36      for (i=0;i<alto;i++) {
37          imagen[i] = (int*)malloc(ancho*sizeof(int));
38      }
39
40      int a,b,c;
41      for(i=0;i<alto;i++){
42          for(j=0;j<ancho;j++){
43              a=fgetc(img1);
44              b=fgetc(img1);
45              c=fgetc(img1);
46
47              if(a==b && b==c){
48                   imagen[i][j]=a;
49              }else{
50                 printf("\nERROR %d = %d = %d",a,b,c);
51              }
52          }
53      }
54
```

```
55      int **imagen2;
56      imagen2 = (int **)malloc(alto*sizeof(int*));
57      for (i=0;i<alto;i++) {
58          imagen2[i] = (int*)malloc(ancho*sizeof(int));
59      }
60      int v,h,k1,k2;
61      for(i=1;i<alto-1;i++){
62          for(j=1;j<ancho-1;j++){
63              v=imagen[i][j]*vertical[1][1]+imagen[i-1][j
                    ]*vertical[0][1]+
64              imagen[i][j-1]*vertical[1][0]+imagen[i-1][j
                    -1]*vertical[0][0]+
65              imagen[i+1][j]*vertical[2][1]+imagen[i][j
                    +1]*vertical[1][2]+
66              imagen[i+1][j+1]*vertical[2][2]+imagen[i-1][
                    j+1]*vertical[0][2]+
67              imagen[i+1][j-1]*vertical[2][0];
68
69              h=imagen[i][j]*horizontal[1][1]+imagen[i-1][
                    j]*horizontal[0][1]+
70              imagen[i][j-1]*horizontal[1][0]+imagen[i-1][
                    j-1]*horizontal[0][0]+
71              imagen[i+1][j]*horizontal[2][1]+imagen[i][j
                    +1]*horizontal[1][2]+
72              imagen[i+1][j+1]*horizontal[2][2]+imagen[i
                    -1][j+1]*horizontal[0][2]+
73              imagen[i+1][j-1]*horizontal[2][0];
74
75              k1=imagen[i][j]*kirch1[1][1]+imagen[i-1][j]*
                    kirch1[0][1]+
76              imagen[i][j-1]*kirch1[1][0]+imagen[i-1][j
                    -1]*kirch1[0][0]+
77              imagen[i+1][j]*kirch1[2][1]+imagen[i][j+1]*
                    kirch1[1][2]+
78              imagen[i+1][j+1]*kirch1[2][2]+imagen[i-1][j
                    +1]*kirch1[0][2]+
79              imagen[i+1][j-1]*kirch1[2][0];
80
81              k2=imagen[i][j]*kirch2[1][1]+imagen[i-1][j]*
                    kirch2[0][1]+
```

```
82              imagen[i][j-1]*kirch2[1][0]+imagen[i-1][j
                    -1]*kirch2[0][0]+
83              imagen[i+1][j]*kirch2[2][1]+imagen[i][j+1]*
                    kirch2[1][2]+
84              imagen[i+1][j+1]*kirch2[2][2]+imagen[i-1][j
                    +1]*kirch2[0][2]+
85              imagen[i+1][j-1]*kirch2[2][0];
86
87              v=abs(v);
88              h=abs(h);
89              k1=abs(k1);
90              k2=abs(k2);
91              if(v>=h && v>=k1 && v>=k2 ){
92                   imagen2[i][j]=v;
93              }
94              if(h>=v && h>=k1 && h>=k2 ){
95                   imagen2[i][j]=h;
96              }
97              if(k1>=h && k1>=v && k1>=k2 ){
98                   imagen2[i][j]=k1;
99              }
100             if(k2>=h && k2>=k1 && k2>=v ){
101                  imagen2[i][j]=v;
102             }
103         }
104     }
105     for(i=0;i<alto;i++){
106         for(j=0;j<ancho;j++){
107             fputc(imagen2[i][j],img2);
108             fputc(imagen2[i][j],img2);
109             fputc(imagen2[i][j],img2);
110         }
111     }
112     fclose(img1);
113     fclose(img2);
114     return 0;
115 }
```