TRATAMIENTO DIGITAL DE IMÁGENES

CON IMtdi

TRATAMIENTO DIGITAL DE IMÁGENES

CON IMtdi

Francisco Guindos Rojas

José Antonio Piedra Fernández

ALMERÍA, 2001

1 Introducción

F. GUINDOS

Cuando hoy hablamos de tratamiento de imágenes, estamos hablando de Informática. Los sistemas informáticos se han convertido en una herramienta indispensable en este campo. Desde las tareas más simples de procesado de imágenes a las más complejas de visión artificial, todas ellas se abordan hoy utilizando los medios que la Informática pone en nuestras manos.

Así, en los estudios universitarios de Informática, se han incluido asignaturas donde el alumno adquiere los conocimientos necesarios para el trabajo en esta área: adquisición, representación y manipulación de imágenes. Se trata de una disciplina donde la realización de prácticas es de suma importancia para su adecuado aprendizaje, pero cuya realización no es fácil de planificar, especialmente cuando se debe limitar a un espacio de tiempo reducido como es un cuatrimestre. El problema surge cuando se pretende realizar una práctica a un cierto nivel en el tratamiento de la imagen, pero para llegar hasta allí hay que realizar toda una serie de tareas previas, que si bien pueden enmarcarse dentro del objetivo de la propia asignatura, reducen el tiempo disponible para el desarrollo de la práctica deseada.

En este manual, con orientación fundamentalmente práctica, se presenta un entorno de trabajo desarrollado por los mismos autores para facilitar la realización de las prácticas en asignaturas de Tratamiento Digital de Imágenes. En los primeros capítulos se hace una introducción al entorno y su uso y en el resto del manual se muestra cada función disponible, incluyendo su fundamento teórico, interfaz de uso y ejemplos de utilización.

No se trata de un entorno de prácticas/trabajo destinado a un usuario final, sino al desarrollador del sistema de Tratamiento de Imágenes, que encontrará en él una valiosa ayuda a la hora de programarlo.

1.1 EL ENTORNO DE TRABAJO IMtdi

La principal dificultad para la realización de prácticas Tratamiento de Imágenes radica en que, antes de poder aplicar cualquier función sobre una imagen es necesario leerla del medio de almacenamiento, almacenarla en una estructura de datos del programa y, frecuentemente, realizar unas tareas de preproceso. Ahí es donde resulta útil este entorno, que ofrece las ayudas necesarias para realizar estas tareas previas con el mínimo esfuerzo. También incluye funciones que facilitan el procesamiento posterior.

Utilizando las herramientas y librerías disponibles en el entorno de trabajo, el alumno podrá crear fácilmente en C++ sus propios programas en los que ponga a prueba las funciones de Tratamiento de Imágenes objeto de interés.

El objetivo fundamental es que el alumno pueda centrarse en el objetivo de la práctica que pretenda desarrollar sin pérdidas de tiempo en la implementación de funciones que no forman parte de los objetivos, pero que son necesarias para poder realizarla. Por ejemplo, el alumno puede realizar una práctica de implementación de filtros sin tener que ocuparse de la lectura o escritura de la imagen.

Por motivos de simplicidad y, dado el hecho de que la mayoría de los algoritmos estudiados trabajan sobre imágenes representadas en niveles de gris, el entorno está especialmente diseñado para manipular este tipo de imágenes y, si bien admite imágenes en color, la cobertura para ellas es reducida.

Distribución de IMtdi

Todo lo necesario para la utilización del entorno de trabajo IMtdi, ficheros de cabecera (*.hpp) y librería (*.lib) se encuentra disponible en la dirección:

http://www.ual.es/~fguindos

2 Digitalización de imágenes

F. GUINDOS

Conocemos el mundo que nos rodea a través de los sentidos, entre los que la vista tiene un papel fundamental. Gracias a ella obtenemos **imágenes** de nuestro entorno que suponen una valiosa información para conocerlo.

Para poder utilizar un ordenador en la manipulación de imágenes, lo primero que hay que hacer es dotarlo de un *ojo* y una forma de representación y almacenamiento de la imagen en su interior.

Una cámara de televisión puede tomar imágenes pero, ¿cómo representarlas de una forma manipulable por un ordenador *digital*? Surge así la necesidad de digitalizar las imágenes obtenidas por las cámaras de visión. Lo mismo ocurre con las imágenes adquiridas mediante cualquier otro sistema, como puede ser un escáner.

En nuestra vida cotidiana estamos acostumbrados a utilizar imágenes de 2 dimensiones, y así lo haremos a lo largo de este manual, aunque la extensión a 3 dimensiones, en caso de ser necesario, es fácil para la mayor parte de lo que aquí aparece. Así, podemos representar una imagen como una función de 2 variables $f(x,y)$ donde x e y son las coordenadas en el espacio bidimensional y los valores de la función representan la información visual de cada punto.

El proceso de **digitalización de una imagen** consiste en convertir la imagen desde un formato continuo en el que existe en el mundo *real* a una representación numérica utilizable por el ordenador. Esta conversión se realiza en dos niveles y para comprenderla hay que conocer algunos fundamentos físicos sobre la luz y composición del color.

2.1 MUESTREO

En primer lugar es necesario discretizar el dominio espacial del la función de imagen *f(x,y)*. A la discretización (y limitación a un intervalo finito) de las coordenadas espaciales *(x,y)* se le denomina **muestreo**.

El muestreo habitualmente se hace de forma que *x* e *y* tomen valores enteros, pero hay algunas diferencias de unos sistemas a otros:

- Hay sistemas que dan valores a *x* e *y* a partir de 0. Otros lo hacen a partir de 1.
- Si bien la *x*, que se destina a la coordenada horizontal, se hace comenzar siempre por la izquierda, la coordenada *y* que se destina a la vertical, en unos sistemas comienza por la parte inferior (la habitual en la representación cartesiana del primer cuadrante) y en otros por la superior (representación habitual para las matrices).

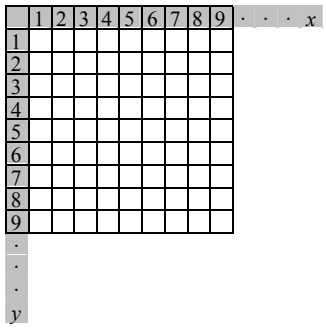


Figura 2–1 Muestreo de la imagen

De esta forma, la función de imagen *f(x,y)* se convierte en una matriz, estructura fácilmente manipulable por un sistema informático.

$$f(x,y)=\begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,M) \\ f(2,1) & f(2,2) & \cdots & f(2,M) \\ \vdots & \vdots & \ddots & \vdots \\ f(N,1) & f(N,2) & \cdots & f(N,M) \end{bmatrix}$$

Figura 2–2 Representación matricial de la imagen

Ahora podemos representar una imagen como un conjunto finito de M × N elementos, a cada uno de los cuales se le denomina **píxel**. En realidad, para cada píxel, se almacenará el valor medio o integral de la imagen en el rectángulo que lo constituye.

Un tema clave para la digitalización de la imagen es la selección de los valores de M y N pues determinarán un factor importante de la calidad de la imagen: la **resolución espacial**. También repercutirán en el espacio necesario para almacenar la imagen.

Figura 2–3 La misma imagen con diferentes muestreos: 300×300, 100×100, 50×50 y 25×25

2.2 CUANTIZACIÓN

El siguiente paso es la digitalización de la amplitud de la función de imagen, lo que se denomina **cuantización**. Para ello se ha de definir una forma de representación numérica adecuada a la información visual aportada por cada píxel. Dos son los tipos de imágenes que se pueden utilizar: de escala de grises o en color.

2.2.1 Imágenes en escala de grises

Cuando la función de imagen mide los valores de intensidad luminosa, nos encontramos con imágenes en escala de grises y se puede representar con un valor numérico de esta magnitud por cada píxel.

Aunque no es estrictamente necesario, sí es la costumbre utilizar números naturales para la cuantización de la intensidad lumínica o brillo. Los motivos fundamentales son dos: el número finito y no muy elevado de niveles producidos por los captadores físicos de imágenes y la considerable economía de almacenamiento de estos números frente a los reales. Además, y con el objetivo de adaptar lo mejor posible la escala al almacenamiento en el ordenador, el número de niveles suele ser una potencia de 2, siendo la más frecuente de $2^8 = 256$ niveles. También resulta interesante la escala de 2 niveles o *blanco y negro*.

Figura 2–4 Una misma imagen con 4 cuantizaciones distintas: 256, 16, 4 y 2 niveles de gris respectivamente

2.2.2 Imágenes en color

La representación del color se hace aprovechando las propiedades ópticas de la composición de la luz. Cualquier color visible por el ojo humano se puede expresar como composición de unos colores básicos.

Cuando la combinación de los colores básicos se hace de forma que al ojo llega la suma de las componentes, nos encontramos con la composición aditiva y la base está formada por los colores rojo, verde y azul (*RGB*). Los tres juntos forman el blanco. Esta es la situación es la que se produce en las pantallas habitualmente utilizadas en los ordenadores.

Figura 2–5 Composición de colores *RGB*

Si la combinación de los colores básicos se hace de forma que cada componente absorbe una parte de la luz blanca que llega hasta ellos, y sólo lo no absorbido llega al ojo del observador, nos encontramos con la composición sustractiva, que es la que ocurre con las tintas de las impresoras en color. La base para este tipo de combinación está formada por los colores cian, magenta y amarillo (*CMY*).

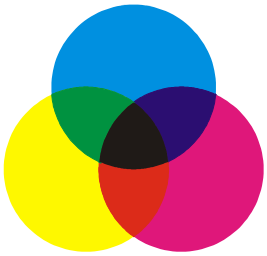


Figura 2–6 Composición de colores *CMY*

No son éstas las únicas bases posibles para definir el espacio del color. Aparte de éstas, y entre las más usadas, se encuentra el modelo de color HSL, que se basa en valores de tono, saturación y brillo. El tono es el color de base. La saturación es la pureza del color o la distancia que lo separa del gris y el brillo es la cantidad de blanco que contiene.

Todos los modelos de color existentes se relacionan entre sí y hay fórmulas que nos permiten pasar de uno a otro. Por ejemplo, entre RGB y CMY:

$$C = MAX - R$$
$$M = MAX - G$$
$$Y = MAX - B$$

Una vez escogido un espacio de color, una imagen se representará indicando el color para cada uno de los píxeles. Para el almacenamiento, el modelo más utilizado es el RGB.

En las imágenes en color también se produce una cuantización, pues cada color se representa con una cierta precisión. De nuevo, se suelen utilizar números naturales para ello y, para adecuarlo al almacenamiento en el ordenador, lo más frecuente es utilizar un byte para cada color de la base, lo que da un total de 3 bytes o 24 bits por píxel. El número de colores diferentes con esta cuantización es de $2^{3 \times 8} = 16.777.215$, muchos más de los que el ojo humano puede distinguir, por lo que se denomina color real. También se utilizan cuantizaciones de 15 ó 16 bits por píxel.

2.3 ALMACENAMIENTO DE LA IMAGEN

Internamente, cada programa almacena las imágenes de la forma que su diseñador ha decidido, aunque lo más frecuente es emplear la estructura matricial ofrecida por el lenguaje de programación utilizado. En un capítulo posterior, veremos cómo lo hace nuestro entorno de programación IMtdi.

Para el almacenamiento en disco, el diseñador de un programa puede elegir entre desarrollar un formato propio o utilizar uno de los formatos estándar existentes, entre los que hay diferencias sustanciales.

El tamaño necesario para almacenar una imagen depende directamente del muestreo y la cuantización. En general, una imagen de M × N píxeles, cada uno de los cuales se representa con b bits necesitará un espacio de M × N × b bits o, lo que es lo mismo, (M × N × b) / 8 bytes.

Sin embargo, el tamaño final de un archivo conteniendo una imagen puede variar de un formato a otro, que se diferencian en varios factores.

2.3.1 Cabecera

Cuando se almacena una imagen, normalmente se guarda con ella información adicional, como son el número de filas y columnas que contiene, cuantización utilizada, resolución a la que fue tomada, etc. Estos datos conforman la **cabecera** de la imagen, que suele encontrarse al principio del fichero.

No hay grandes diferencias de un formato a otro en cuanto a los datos que forman la cabecera, pero sí en cuanto a la estructura, que es causa fundamental de incompatibilidades entre ellos.

2.3.2 Paleta de colores

El espacio necesario para almacenar un píxel no es mucho, pero si tenemos en cuenta la gran cantidad de píxeles que puede tener una imagen, puede resultar que el tamaño final resulte en ciertos casos excesivo. Por ejemplo, una imagen típica en color de 1.000 × 1.000 píxeles ocupa aproximadamente 3 Mbytes.

Una estrategia para reducir el tamaño es la utilización de una paleta de color. Los colores disponibles para una imagen serán un subconjunto del espacio completo, normalmente escogidos entre los más usados en dicha imagen. De esta forma, y siempre que se incluya la paleta en el fichero de la imagen, para especificar el color de un píxel se indica el índice de la paleta donde se encuentra ese color. Nos encontramos así con imágenes definidas en un espacio de color amplio, pero del que sólo usan un número reducido de colores.

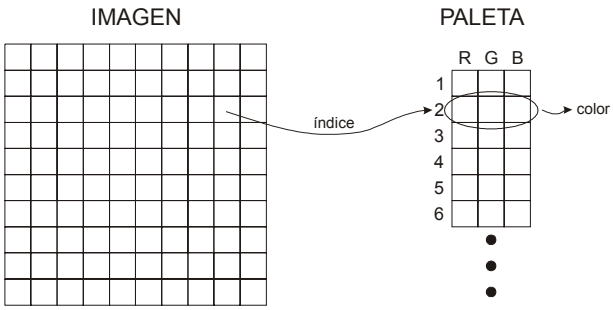


Figura 2–7 Determinación del color a través de la paleta

Esto se hace en formatos para almacenamiento de imágenes, pero también se hace en el almacenamiento interno de algunos programas o sistemas gráficos y, a nivel de hardware, en algunos modos de visualización en la pantalla del ordenador.

Así, es frecuente encontrar imágenes con una paleta de 256 colores definida en un espacio de 16 millones de colores (**color real**) que ocupan poco más de la tercera parte que el original.

2.3.3 Compresión

Otra posibilidad para reducir el tamaño ocupado por una imagen es la utilización de técnicas de compresión en la propia especificación del formato de almacenamiento. En este aspecto hay que diferenciar dos tipos de compresión.

Por un lado, se utilizan técnicas desarrolladas mediante teoría de información y aplicadas para comprimir cualquier tipo de datos y con las que la información, una vez restaurada es exactamente igual que la original. De este tipo, las compresiones más utilizadas son LZW y RLE.

También se utilizan métodos de compresión diseñados específicamente para imágenes, que producen alteraciones en la imagen, a veces inapreciables, pero que a costa de ello pueden conseguir tasas de compresión espectaculares. Es el caso de la compresión JFIF usada en el formato JPEG.

2.4 FUENTES DE INFORMACIÓN

- ***http://www.wotsit.org***

  En este servidor se pueden encontrar las especificaciones de prácticamente cualquier formato gráfico y código fuente para leer o escribir buena parte de ellos.

3 Matrices

F. GUINDOS

3.1 CONCEPTOS GENERALES

Como se ha visto anteriormente, la estructura básica para la representación de una imagen es la matriz. No es, por tanto, de extrañar que el módulo más importante para la programación en el entorno IMtdi sea el dedicado a las matrices. Su diseño se ha hecho de forma que mejora notablemente las capacidades básicas de manipulación de estas estructuras presente en el lenguaje de programación utilizado, C++.

Este módulo, implementado como una clase de C++ denominada ***C_Matrix*** permite definir y manipular matrices de forma cómoda y segura. Con *C_Matrix* se pueden definir matrices de números reales, acceder a sus elementos, obtener o modificar sus características, etc. También se incluye una buena cantidad de funciones, algunas de ellas claramente orientadas al Tratamiento Digital de Imágenes.

Una matriz es un conjunto de números organizados en una retícula rectangular de filas y columnas. La Figura 3–1 muestra la representación de una matriz *A*, de N filas × M columnas.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,M} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,M} \end{bmatrix}$$

Figura 3–1 Representación matricial

3.1.1 Índices

Las filas y columnas de una matriz se numeran con **índices**. El modo habitual de hacerlo es comenzando por la esquina superior izquierda y empezando por el número 1.

Sin embargo, en ciertas ocasiones, es más interesante numerar de otras formas; comenzando por 0 ó de –n a +n, por ejemplo.

En C++, los índices de las matrices siempre comienzan por 0. La primera ventaja de clase C_Matrix es que permite escoger al programador el comienzo y fin de la numeración, tanto para filas como para columnas.

Otra ventaja que aporta esta clase en la manipulación de índices es la protección frente a un error frecuente y desastroso cuando se programa en C++: los índices fuera de rango. Cuando esto ocurre, los programas suelen quedarse "colgados"; a menudo, tiempo después de haberse producido el fallo, lo que hace que sean difíciles de detectar. Con la clase *C_Matrix*, si se produce alguno de estos accesos, se genera un mensaje de error inmediatamente y se impide dicho acceso, por lo que el programador puede conocer fácilmente el punto donde falla el programa, que no se colgará.

3.1.2 Matrices y submatrices

Es frecuente que surja la necesidad de acceder a una zona reducida de la matriz para realizar alguna operación en ella. Cuando esta zona es rectangular, y tiene por tanto estructura de matriz, se dice que **submatriz** de la anterior. La clase C_Matrix incluye la posibilidad de declarar una nueva matriz que sea submatriz de otra creada previamente (supermatriz).

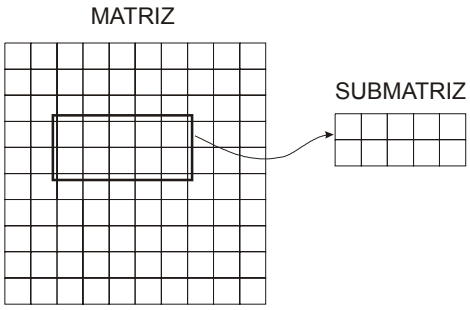


Figura 3–2 Submatriz

Una submatriz no es copia de la zona correspondiente de la supermatriz, sino que supone una forma de acceso a los mismos datos. Es decir, que una alteración en un dato de la submatriz afecta directamente al dato correspondiente en la supermatriz, y viceversa.

Una vez definida una submatriz, se pueden aplicar a ella todas las funciones definidas para matrices (incluso crear una submatriz de la submatriz).

3.1.3 Vecindad

Algunas de los métodos definidos sobre las matrices utilizan en concepto de vecindad. Son dos los tipos de vecindad utilizados en la retícula bidimensional definida por una matriz:

4-vecindad y 8-vecindad. Los métodos cuyo funcionamiento depende del tipo de vecindad utilizado admiten un parámetro en el que es especifica cuál de ellas ha de usar.

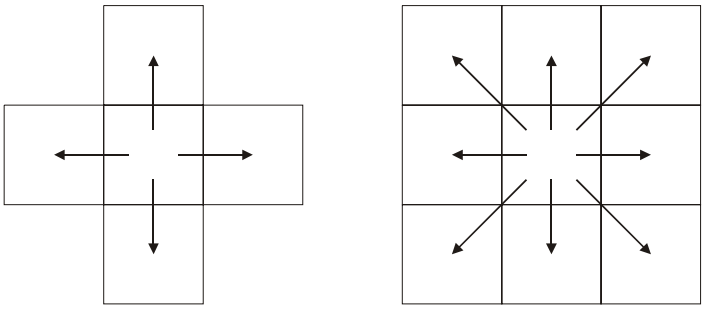


Figura 3–3 4-vecindad y 8-vecindad en retículas bidimensionales

3.2 MANIPULACIÓN BÁSICA DE MATRICES

Para utilizar el módulo con la clase C_Matrix se incluye el fichero de cabecera C_Matrix.hpp:

```
#include <C_Matrix.hpp>
```

3.2.1 Tipos de datos

Los tipos de datos definidos en la clase son:

ElementT

El tipo *ElementT* es el que se utiliza interiormente en toda la clase para los elementos de la matriz y se puede usar también para declarar datos independientes que hayan de ser del mismo tipo. Se halla definido como real de doble precisión (*double*).

IndexT

Es el tipo utilizado en los índices de la matriz: inicio y fin de filas y columnas y variables para acceso a sus elementos.

3.2.2 Constructores

Son varios los constructores de que dispone la clase. Todos ellos permiten crear un objeto que pertenezca a ella.

C_Matrix

C_Matrix ( )

Crea una matriz vacía. Se usa para crear una matriz cuyo contenido vendrá dado por otra función, por ejemplo, cuando se vaya a leer de un fichero.

Ejemplo

```
C_Matrix matriz;
```

C_Matrix

C_Matrix (const IndexT firstRow, const IndexT lastRow, const IndexT firstCol, const IndexT lastCol, const ElementT initValue = 0)

Es el constructor usual. Se especifica el tamaño (índices inicial y final) y valores iniciales.

Parámetros de entrada

firstRow Índice de la primera fila.

lastRow Índice de la última fila. Ha de ser mayor o igual que *firstRow*.

firstCol Índice de la primera columna.

lastCol Índice de la última columna. Ha de ser mayor o igual que *firstCol*.

initValue Valor inicial para cada elemento. Por omisión, se inicializarán a 0.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5, 9);
```

Crea una matriz con filas y columnas de 1 a 5 y con un valor inicial en cada elemento de 9.

	1	2	3	4	5
1	9	9	9	9	9
2	9	9	9	9	9
3	9	9	9	9	9
4	9	9	9	9	9
5	9	9	9	9	9

Figura 3–4 C_Matrix matriz (1, 5, 1, 5, 9)

C_Matrix

C_Matrix (C_Matrix & superMatrix, const IndexT firstRow, const IndexT lastRow, const IndexT firstCol, const IndexT lastCol, const IndexT firstRowSuperMat, const IndexT firstColSuperMat)

Crea una **submatriz** de una matriz creada previamente. Cuando se crea una matriz, los índices de ésta no tienen necesariamente que coincidir con los equivalentes en la supermatriz. Por ejemplo, para hacer una convolución, se puede definir una submatriz con índices de –1 a +1, independientemente de donde se ubique en la supermatriz. Hay, por tanto, que indicar sus índices inicial y final para filas y columnas y la posición donde se ubica en la supermatriz (fila y columna de la esquina superior izquierda).

Parámetros de entrada

superMatrix Matriz de la que se va a crear una submatriz.

firstRow Índice de la primera fila de la submatriz.

lastRow Índice de la última fila de la submatriz. Ha de ser mayor o igual que firstRow.

firstCol Índice de la primera columna de la submatriz.

lastCol Índice de la última columna de la submatriz. Ha de ser mayor o igual que firstCol.

firstRowSuperMat Fila de la matriz donde comienza la submatriz

firstColSuperMat Columna de la matriz donde comienza la submatriz

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);
C_Matrix submatriz (matriz, -1, 1, -1, 1, 1, 2);
```

Tras crear una matriz de 8 × 8, crea una submatriz de índices –1 a 1para filas y columnas, ubicado con la esquina superior izquierda en la fila 2 y columna 4.

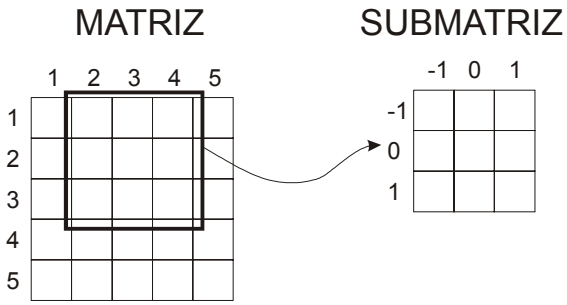


Figura 3–5 Creación de una submatriz

3.2.3 Destructores

Como cualquier objeto en C++, cuando una matriz ya no es necesaria, se puede destruir, liberando así la memoria ocupada.

Es importante tener en cuenta que no se puede destruir una matriz de la que se encuentren definidas submatrices. Si es necesario, primero se destruirán las submatrices y posteriormente la supermatriz.

~C_Matrix

~C_Matrix ( )

Cuando se destruye una matriz, se libera tanto el espacio ocupado por sus elementos (el cuerpo de la matriz) como por los datos que definen su estructura como objeto.

Free

Free ( )

Libera el espacio ocupado por la matriz, pero conserva el objeto. No es un destructor propiamente del lenguaje C++, pero se incluye porque en algunos casos puede resultar adecuado liberar sólo el cuerpo de la matriz por motivos de ahorro de memoria, pero mantener el objeto para usarlo posteriormente.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);
. . .
matriz.Free();
```

3.2.4 Acceso a datos sobre la matriz

Todos los datos sobre la estructura de la clase están protegidos mediante el uso de la cláusula *private*, pero hay funciones para acceder a todos aquellos de interés.

Empty

bool Empty ( )

Indica si una matriz está vacía. Es decir, si se ha creado con el constructor C_Matrix ( ) o se trata de una matriz cuyo cuerpo se ha eliminado con *Free ( )*.

Ejemplo

```
C_Matrix matriz;
```

```
. . .
if ( matriz.Empty() ) { printf("Matriz vacía\n"); }
```

FirstRow

IndexT FirstRow ( )

Devuelve el índice de la primera fila de la matriz.

Ejemplo más abajo

LastRow

IndexT LastRow ( )

Devuelve el índice de la última fila de la matriz.

Ejemplo más abajo

FirstCol

IndexT FirstCol ( )

Devuelve el índice de la primera columna de la matriz.

Ejemplo más abajo

LastCol

IndexT LastCol ( )

Devuelve el índice de la última columna de la matriz.

Ejemplo más abajo

RowN

IndexT RowN ( )

Devuelve el número de filas de la matriz.

ColN

IndexT ColN ( )

Devuelve el número de columnas de la matriz.

Fail

bool Fail ( )

Cuando se produce un fallo en la ejecución de un método de una matriz, además de producir el correspondiente aviso del error, se activa un indicador que permite, dentro del programa, conocer esta condición. El método *Fail ( )* devuelve esta condición de fallo. *true* si ha habido un error con esta matriz y *false* en caso contrario.

Ejemplo más abajo

SetFail

SetFail ( )

Este método activa el indicador de fallo de una matriz. Útil cuando una operación sobre ella ha fallado y, por tanto, los datos que contenga pueden no ser válidos. El valor de este indicador de fallo se obtiene con el método *Fail ( )*.

Ejemplo más abajo

Clear

Clear ( )

Cuando se activa el indicador de fallo en una matriz, permanece hasta que se desactive, lo que se hace con este método.

Ejemplo para FirstRow, LastRow, FirstCol, LastCol, SetFail, Fail, Clear

```
C_Matrix::IndexT fila, col;
C_Matrix matriz (1, 5, 1, 5);

for (fila = matriz.FirstRow(); fila <= matriz.LastRow(); fila++)
  for (col = matriz.FirstCol(); col <= matriz.LastCol(); col++)
  {
    . . .
    if (<fallo>) { matriz.SetFail();}
    . . .
  }
if (matriz.Fail())
{
  printf (”Ha habido algun fallo\n”);
  matriz.Clear();
}
```

3.2.5 Acceso a los elementos de la matriz

Para el acceso a cada elemento de una matriz se ha sobrecargado el operador ( ), lo que permite escribir de la forma más natural posible. Este operador es válido tanto para obtener el valor un elemento como para modificarlo.

El método está protegido frente a accesos con índices fuera de la matriz. Si se accede a un elemento inexistente, si es para lectura, devuelve un valor no definido y activa el indicador de fallo de la matriz. Si ocurre al intentar modificar un elemento, simplemente se activa el indicador de fallo. En ambos casos, el programa puede seguir funcionando normalmente. Es responsabilidad del programador comprobar la condición de fallo y actuar en consecuencia.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

dato = matriz(2, 3);
. . .
matriz(1, 5) = 3;
```

3.2.6 Otros métodos

Hay algunos métodos más para manipulación básica de matrices. Los más notables son:

Operador =

El operador = se ha sobrecargado para poder realizar la asignación del valor de una matriz a otra.

Ejemplo

```
C_Matrix A(1, 5, 1, 5, 0);
C_Matrix B();
. . .
B = A;
```

Reindex

Reindex (const IndexT newFirstRow, const IndexT newFirstCol)

Cambia los índices de una matriz. Los datos contenidos se mantienen fijos respecto a la matriz completa.

Ejemplo

```
C_Matrix matriz (0, 4, 0, 4);

. . .
matriz.Reindex(1, 1);
```

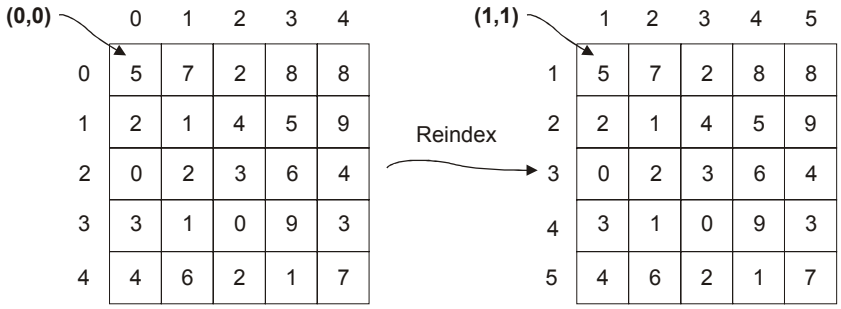


Figura 3–6 Reindex: Cambio de índices

Resize

Resize (const IndexT newFirstRow, const IndexT newLastRow, const IndexT newFirstCol, const IndexT newLastCol, const ElementT defValue = 0)

Cambia el tamaño de una matriz, lo que supone una modificación en los índices. El valor de cada elemento se mantiene fijo respecto a los índices (en aquellos que se conserven). Permite especificar el valor que tendrán los elementos para los nuevos índices.

Ejemplo

```
C_Matrix matriz (0, 4, 0, 4);
. . .
matriz.Resize (1, 6, 1, 6, 0);
```

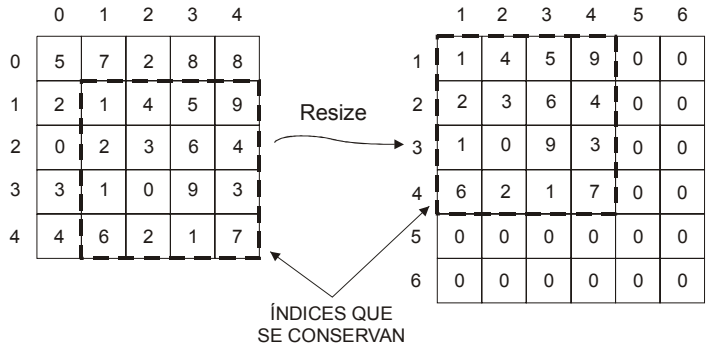


Figura 3–7 Resize: sólo se mantienen los datos para los índices que se conservan

MoveSubMat

MoveSubMat (const IndexT newFirstRowSuperMat, const IndexT newFirstColSuperMat)

Con este método podemos reubicar una matriz dentro de la supermatriz. Ningún elemento se ve alterado por ello.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);
C_Matrix submatriz (matriz, -1, 1, -1, 1, 1, 2);

. . .
submatriz.MoveSubMat(2,3);
```

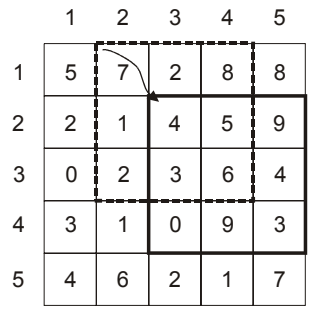


Figura 3–8 MoveSubMat cambia de sitio una submatriz dentro de la matriz

3.3 FUNCIONES ESTADÍSTICAS

Se incluyen algunas funciones para realizar funciones estadísticas simples sobre los elementos de la matriz:

Min

ElementT Min ( )

Devuelve el mínimo de los valores en la matriz.

Ejemplo más abajo

Max

ElementT Max ( )

Devuelve el máximo de los valores en la matriz.

Ejemplo más abajo

Sum

ElementT Sum ( )

Devuelve la suma de todos los elementos de la matriz.

Ejemplo más abajo

Mean

ElementT Mean ( )

Devuelve la media aritmética de los elementos de la matriz.

Ejemplo más abajo

Mode

ElementT Mode (long int * frequency = NULL, ElementT discardValue = C_MATRIX_NAN)

Calcula y devuelve la moda de los elementos de la matriz.

Parámetros de entrada

discardValue (opcional) Es un valor que se debe ignorar en el cálculo. Los elementos con ese valor no se tendrán en cuenta para el cálculo de la moda

Parámetros de salida

frequency (opcional) Si se incluye este parámetro, en él se devolverá el número de veces que aparece el valor de la moda (**frecuencia**).

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);
C_Matrix::ElementT min, max, sum, mean, mode;
long int frec;
. . .
min = matriz.Min();
max = matriz.Max();
sum = matriz.Sum();
mean = matriz.Mean();
mode = matriz.Mode(&frec);
```

	1	2	3	4	5
1	3	7	6	7	3
2	9	0	4	6	5
3	4	1	4	1	2
4	9	4	5	1	7
5	9	8	8	4	9

Figura 3–9 Matriz para aplicar los cálculos estadísticos

```
Min  =   0    Max  =   9
Sum  = 126    Mean =   5.04
Mode =   4    Frec =   5
```

3.4 OPERACIONES CON LOS ELEMENTOS DE LA MATRIZ

3.4.1 Asignación de valores

SetValue

SetValue (const ElementT newValue)

Asigna el valor *newValue* a todos los elementos de la matriz.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

matriz.SetValue(9);
```

	1	2	3	4	5
1	9	9	9	9	9
2	9	9	9	9	9
3	9	9	9	9	9
4	9	9	9	9	9
5	9	9	9	9	9

Figura 3–10 matriz.SetValue(9)

SetValue

SetValue (const ElementT oldValue, const ElementT newValue)

Asigna el valor *newValue* a todos los elementos de la matriz cuyo valor sea *oldValue*.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.SetValue(0, 9);
```

	1	2	3	4	5
1	1	0	1	1	0
2	0	1	0	1	0
3	0	1	0	1	1
4	1	0	1	1	1
5	0	0	1	1	1

Antes

	1	2	3	4	5
1	1	9	1	1	9
2	9	1	9	1	9
3	9	1	9	1	1
4	1	9	1	1	1
5	9	9	1	1	1

Después

Figura 3–11 Antes y después de ejecutar matriz.SetValue(0, 9)

SetValue

SetValue (const ElementT oldValueMin, const ElementT oldValueMax, const ElementT newValue)

Asigna el valor *newValue* a todos los elementos de la matriz cuyo valor esté comprendido entre *oldValueMin* y *oldValueMax*.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.SetValue(1, 5, 0);
```

	1	2	3	4	5
1	7	6	6	6	2
2	4	3	9	2	1
3	1	5	2	7	7
4	9	7	5	3	5
5	5	9	1	6	2

Antes

	1	2	3	4	5
1	7	6	6	6	0
2	0	0	9	0	0
3	0	0	0	7	7
4	9	7	0	0	0
5	0	9	0	6	0

Después

Figura 3–12 Antes y después de ejecutar matriz.SetValue(1, 5, 0)

Serie

Serie (const ElementT initValue, const ElementT rowInc, const ElementT colInc)

Crea una serie aritmética en la matriz.

Parámetros de entrada

initValue Valor para el elemento en la primera fila, primera columna; a partir del cual comenzará la serie.

rowInc. Incremento que se produce de un elemento al elemento en la fila siguiente.

colInc. Incremento que se produce de un elemento al elemento en la columna siguiente.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);
matriz.Serie (11, 10, 1);
```

	1	2	3	4	5
1	11	12	13	14	15
2	21	22	23	24	25
3	31	32	33	34	35
4	41	42	43	44	45
5	51	52	53	54	55

Figura 3–13 matriz.Serie (11, 10, 1)

3.4.2 Limitación de valores

Trunc

Trunc (const ElementT min, const ElementT max)

Realiza un truncado de los valores de los elementos de la matriz. Cualquier valor por debajo de *min* se cambia por *min* y los valores por encima de *max* se cambian por *max*.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.Trunc(10, 20);
```

	1	2	3	4	5
1	9	27	19	26	3
2	1	5	2	12	9
3	30	1	4	7	1
4	15	17	12	2	1
5	22	18	20	20	11

Antes

	1	2	3	4	5
1	10	20	19	20	10
2	10	10	10	12	10
3	20	10	10	10	10
4	15	17	12	10	10
5	20	18	20	20	11

Después

Figura 3–14 Antes y después de ejecutar matriz.Trunc(10, 20)

Stretch

Stretch (const ElementT min, const ElementT max)

Si el rango de valores de la matriz sobrepasa el formado por [*min*, *max*], el rango de la matriz se comprime hasta que esté incluido en [*min*, *max*].

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.Stretch(10, 20);
```

	1	2	3	4	5
1	21	5	5	17	13
2	23	23	8	14	11
3	18	22	18	18	8
4	5	21	15	3	28
5	22	24	17	20	16

Antes

	1	2	3	4	5
1	17.2	10.8	10.8	15.6	14
2	18	18	12	14.4	13.2
3	16	17.6	16	16	12
4	10.8	17.2	14.8	10	20
5	17.6	18.4	15.6	16.8	15.2

Después

Figura 3–15 Antes y después de ejecutar matriz.Stretch(10, 20)

3.4.3 Operaciones matemáticas

Abs

Abs ( )

Cambia cada elemento de la matriz por su valor absoluto.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.Abs();
```

	1	2	3	4	5
1	5	-5	2	-8	2
2	-2	-9	-4	5	-2
3	1	2	6	-5	0
4	6	-6	8	6	3
5	-9	3	8	1	7

Antes

	1	2	3	4	5
1	5	5	2	8	2
2	2	9	4	5	2
3	1	2	6	5	0
4	6	6	8	6	3
5	9	3	8	1	7

Después

Figura 3–16 Antes y después de ejecutar matriz.Abs ()

Add

Add (C_Matrix & mat1, C_Matrix & mat2)

Efectúa la suma matricial de *mat1* + *mat2* y almacena el resultado en la matriz para la que se ejecuta el método.

Ejemplo

```
C_Matrix matriz1 (1, 5, 1, 5);
C_Matrix matriz2 (1, 5, 1, 5);
C_Matrix matriz3;

. . .
matriz3.Add(matriz1, matriz2);
```

	1	2	3	4	5
1	1	7	7	5	8
2	1	10	2	3	7
3	6	7	9	8	8
4	3	9	2	8	2
5	5	4	4	3	7

matriz1

	1	2	3	4	5
1	10	4	6	8	5
2	10	4	8	7	3
3	10	4	4	2	6
4	9	4	9	10	2
5	2	5	8	4	1

matriz2

	1	2	3	4	5
1	11	11	13	13	13
2	11	14	10	10	10
3	16	11	13	10	14
4	12	13	11	18	4
5	7	9	12	7	8

matriz3

Figura 3–17 matriz3.Add(matriz1, matriz2)

Subtract

Subtract (C_Matrix & mat1, C_Matrix & mat2)

Efectúa la sustracción matricial de *mat1* - *mat2* y almacena el resultado en la matriz para la que se ejecuta el método.

Ejemplo

```
C_Matrix matriz1 (1, 5, 1, 5);
C_Matrix matriz2 (1, 5, 1, 5);
C_Matrix matriz3;

. . .
matriz3. Subtract(matriz1, matriz2);
```

	1	2	3	4	5
1	6	2	7	9	4
2	3	4	7	6	7
3	9	6	3	10	7
4	6	3	1	7	9
5	3	6	7	6	4

matriz1

	1	2	3	4	5
1	4	2	10	5	6
2	8	5	2	5	1
3	9	3	6	1	5
4	10	1	3	3	3
5	8	9	8	8	3

matriz2

	1	2	3	4	5
1	2	0	-3	4	-2
2	-5	-1	5	1	6
3	0	3	-3	9	2
4	-4	2	-2	4	6
5	-5	-3	-1	-2	1

matriz3

Figura 3–18 matriz3.Subtract(matriz1, matriz2)

MultiplyElm

MultiplyElm (C_Matrix & mat1, C_Matrix & mat2)

Multiplica uno a uno cada elemento de *mat1* por el correspondiente de *mat2* y almacena el resultado en la matriz para la que se ejecuta el método. No se trata de una multiplicación matricial.

Ejemplo

```
C_Matrix matriz1 (1, 5, 1, 5);
C_Matrix matriz2 (1, 5, 1, 5);
C_Matrix matriz3;

. . .
matriz3. MultiplyElm(matriz1, matriz2);
```

	1	2	3	4	5
1	1	2	6	6	5
2	6	7	8	4	3
3	1	5	1	7	3
4	3	4	8	2	5
5	6	1	6	4	3

matriz1

	1	2	3	4	5
1	7	5	9	3	8
2	4	1	8	8	6
3	7	7	2	4	2
4	8	3	2	5	3
5	3	2	7	7	6

matriz2

	1	2	3	4	5
1	7	10	54	18	40
2	24	7	64	32	18
3	7	35	2	28	6
4	24	12	16	10	15
5	18	2	42	28	18

matriz3

Figura 3–19 matriz3.MultiplyElm(matriz1, matriz2)

DivideElm

DivideElm (C_Matrix & mat1, C_Matrix & mat2)

Divide uno a uno cada elemento de *mat1* por el correspondiente de *mat2* y almacena el resultado en la matriz para la que se ejecuta el método. No se trata de una división matricial.

Ejemplo

```
C_Matrix matriz1 (1, 5, 1, 5);
C_Matrix matriz2 (1, 5, 1, 5);
C_Matrix matriz3;

. . .
matriz3.DivideElm(matriz1, matriz2);
```

	1	2	3	4	5
1	3	9	2	1	10
2	7	3	4	7	8
3	8	9	4	7	6
4	7	5	1	5	3
5	4	9	6	5	7

matriz1

	1	2	3	4	5
1	10	9	9	6	10
2	2	1	7	6	5
3	7	8	3	1	2
4	1	9	3	7	5
5	9	10	6	3	3

matriz2

	1	2	3	4	5
1	0.3	1	0.2	0.2	1
2	3.5	3	0.6	1.2	1.6
3	1.1	1.1	1.3	7	3
4	7	0.6	0.3	0.7	0.6
5	0.4	0.9	1	1.7	2.3

matriz3

Figura 3–20 matriz3.DivideElm(matriz1, matriz2)

AddEscalar

AddEscalar (ElementT escalar)

Suma un valor escalar a cada elemento de la matriz.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.AddEscalar(10);
```

	1	2	3	4	5
1	0	4	6	6	1
2	7	4	6	6	9
3	5	2	3	7	5
4	4	4	0	4	3
5	9	7	0	9	4

Antes

	1	2	3	4	5
1	10	14	16	16	11
2	17	14	16	16	19
3	15	12	13	17	15
4	14	14	10	14	13
5	19	17	10	19	14

Después

Figura 3–21 Antes y después de ejecutar matriz.AddEscalar()

SubtractEscalar

SubtractEscalar (ElementT escalar)

Resta un valor escalar a cada elemento de la matriz.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.SubtractEscalar(10);
```

	1	2	3	4	5
1	10	12	13	19	13
2	15	15	16	13	19
3	14	18	10	14	19
4	18	18	19	14	12
5	10	14	13	12	19

Antes

	1	2	3	4	5
1	0	2	3	9	3
2	5	5	6	3	9
3	4	8	0	4	9
4	8	8	9	4	2
5	0	4	3	2	9

Después

Figura 3–22 Antes y después de ejecutar matriz.SubtractEscalar()

SubtractFromEscalar

SubtractFromEscalar (ElementT escalar)

Resta cada elemento de la matriz de un valor escalar.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.SubtractFromEscalar(10);
```

	1	2	3	4	5
1	7	7	6	9	6
2	1	1	3	6	1
3	10	4	7	1	7
4	8	7	5	3	8
5	7	7	6	3	3

Antes

	1	2	3	4	5
1	3	3	4	1	4
2	9	9	7	4	9
3	0	6	3	9	3
4	2	3	5	7	2
5	3	3	4	7	7

Después

Figura 3–23 Antes y después de ejecutar matriz.SubtractFromEscalar()

MultiplyEscalar

MultiplyEscalar (ElementT escalar)

Multiplica por un valor escalar cada elemento de la matriz.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.MultiplyEscalar(10);
```

	1	2	3	4	5
1	8	7	3	7	6
2	7	4	0	7	8
3	4	3	2	6	4
4	9	4	6	1	4
5	8	4	6	9	5

Antes

	1	2	3	4	5
1	80	70	30	70	60
2	70	40	0	70	80
3	40	30	20	60	40
4	90	40	60	10	40
5	80	40	60	90	50

Después

Figura 3–24 Antes y después de ejecutar matriz.MultiplyEscalar()

DivideEscalar

DivideEscalar (ElementT escalar)

Divide por un valor escalar cada elemento de la matriz.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.DivideEscalar(10);
```

	1	2	3	4	5
1	3	7	4	8	8
2	2	1	7	7	8
3	3	8	9	0	3
4	3	1	3	4	0
5	3	0	0	3	5

Antes

	1	2	3	4	5
1	0.3	0.7	0.4	0.8	0.8
2	0.2	0.1	0.7	0.7	0.8
3	0.3	0.8	0.9	0	0.3
4	0.3	0.1	0.3	0.4	0
5	0.3	0	0	0.3	0.5

Después

Figura 3–25 Antes y después de ejecutar matriz.DivideEscalar()

3.5 FUNCIONES DE PROPÓSITO ESPECÍFICO

Se incluyen también algunas funciones de utilidad bastante más específica ideadas fundamentalmente para realizar un tratamiento de imágenes, pero que sin embargo, operan a nivel de matriz, por lo que se incluyen en la clase *C_Matrix*.

Gaussian

Gaussian (const float std)

Da valores a los elementos de una matriz siguiendo una distribución gausiana. La matriz ha de estar creada y dimensionada previamente. El resultado final se normaliza de forma que la suma de todos los elementos sea 1.

Parámetros de entrada

std Desviación estándar de la distribución.

Ejemplo

```
C_Matrix matriz (-2, 2, -2, 2);
```

```
matriz.Gaussian(0.5);
```

	-2	-1	0	1	2
-2	0.0013	0.0085	0.016	0.0085	0.0013
-1	0.0085	0.057	0.11	0.057	0.0085
0	0.016	0.11	0.2	0.11	0.016
1	0.0085	0.057	0.11	0.057	0.0085
2	0.0013	0.0085	0.016	0.0085	0.0013

Figura 3–26 matriz.Gaussian(0.5);

Convolution

Convolution (C_Matrix & matrix, C_Matrix & convMatrix)

Realiza una convolución. El resultado se almacena en la matriz para la que se ejecuta el método.

En las filas y columnas más exteriores, donde no se puede colocar la matriz de convolución, el resultado es 0.

Parámetros de entrada

matrix Matriz original. No puede coincidir con aquella para la que se ejecuta el método.

convMatrix Matriz de convolución. Ha de tener un número impar de filas y columnas.

Ejemplo

```
C_Matrix matriz1 (1, 10, 1, 10);
C_Matrix matriz2 (-1, 1, -1, 1);
C_Matrix matriz3;

. . .
matriz3.Convolution(matriz1, matriz2);
```

	1	2	3	4	5	6	7	8	9	10
1	6	3	10	3	6	9	2	3	5	8
2	5	1	1	5	7	8	7	1	4	4
3	1	6	9	10	6	3	9	7	9	2
4	3	8	9	7	7	10	1	10	7	9
5	5	6	10	4	10	9	9	3	1	10
6	1	9	3	7	5	9	10	6	3	3
7	1	5	7	7	2	8	5	7	7	10
8	6	3	4	8	6	5	5	1	5	4
9	10	8	1	10	5	1	3	4	10	4
10	6	6	7	4	10	5	9	1	5	10

matriz1

	-1	0	1
-1	0.00033	0.017	0.00033
0	0.017	0.93	0.017
1	0.00033	0.017	0.00033

matriz2

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	1.2	1.4	5	6.9	7.9	6.9	1.3	4.1	0
3	0	5.9	8.8	9.8	6.1	3.4	8.7	7	8.7	0
4	0	7.9	9	7	7.1	9.7	1.6	9.6	7	0
5	0	6.1	9.7	4.3	9.7	9	8.6	3.2	1.4	0
6	0	8.5	3.3	6.9	5.1	8.9	9.8	6	3.1	0
7	0	5	6.8	6.9	2.3	7.8	5.2	6.8	6.9	0
8	0	3.2	4.1	7.9	5.9	5	4.9	1.3	5	0
9	0	7.8	1.4	9.6	5.1	1.3	3.1	4	9.6	0
10	0	0	0	0	0	0	0	0	0	0

matriz3

Figura 3–27 matriz3.Convolution(matriz1, matriz2)

3.6 MÉTODOS DE ENTRADA Y SALIDA

Print

Print (int colWidth, int maxDigits)

Muestra en pantalla una matriz.

Parámetros de entrada

colWidth Ancho de la columna.

maxDigits Número máximo de dígitos significativos.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.Print(3, 1);
```

```
                1   2   3   4   5
              ---------------------
            1|   4   3   9   2   0
            2|   8   3   5   7   7
            3|   0   5   5   2   0
            4|   9   1   3   3   9
            5|   4   4   0   2   4
```

Figura 3–28 matriz.Print(3, 1)

Write

Write (const char * fileName)

Escribe una matriz en un fichero de texto. En la primera escribe el número de filas y de columnas de la matriz.

Parámetros de entrada

fileName Nombre del fichero.

Ejemplo

```
C_Matrix matriz (1, 5, 1, 5);

. . .
matriz.Write("Matriz.txt");
```

Read

Read (const char * fileName)

Lee una matriz de un fichero de texto. La primera línea debe tener el número de filas y de columnas.

Parámetros de entrada

fileName Nombre del fichero.

Ejemplo

```
C_Matrix matriz;

. . .
matriz.Read("Matriz.txt");
```

4 Imágenes

F. GUINDOS

4.1 CONCEPTOS GENERALES

Como se vio en el capítulo de digitalización de imágenes, una imagen representada utilizando una paleta es una matriz de índices que hacen referencia a los elementos de dicha paleta. A su vez, la paleta es otra matriz, en la que cada columna representa el valor para un componente de color y cada fila es un color utilizado por la imagen.

En el entorno de programación IM-tdi se ha incluido un módulo para manipulación de imágenes. En él se encuentra definida la clase ***C_Image*** que contiene los datos y métodos necesarios para el trabajo con imágenes con paleta.

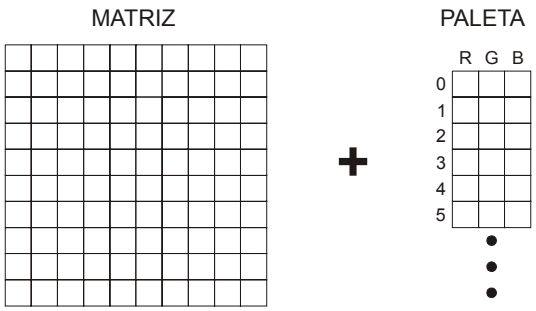


Figura 4–1Imagen = matriz + paleta

La clase *C_Image* se ha definido como clase derivada de *C_Matrix* por lo que hereda todos los métodos definidos para las matrices. Así, si aplicamos cualquier método de la clase *C_Matrix* sobre una imagen, operará sobre la matriz principal de ésta. Además, se incluyen algunos datos específicos de una imagen, como es la presencia de la paleta o los métodos de lectura y escritura de formatos gráficos.

La paleta también es una matriz, pero para utilizar sobre ella los métodos definidos para matrices, hay que hacer referencia explícita a ella.

4.2 MANIPULACIÓN BÁSICA DE IMÁGENES

Para utilizar el módulo con la clase *C_Image* se incluye el fichero de cabecera *C_Image.hpp*:

```
#include <C_Image.hpp>
```

4.2.1 Constantes definidas

Las constantes definidas en este módulo son:

C_RED

Columna de la paleta para la componente roja.

C_GREEN

Columna de la paleta para la componente verde.

C_BLUE

Columna de la paleta para la componente azul.

4.2.2 Datos

Además de aquellos datos heredados de la clase *C_Matrix*, en la clase *C_Image* se incluye:

Palette

Paleta de la imagen. Es una matriz de 3 columnas con índices (*C_BLUE*, *C_GREEN* y *C_RED*) y tantas filas como colores, siendo su primer índice 0.

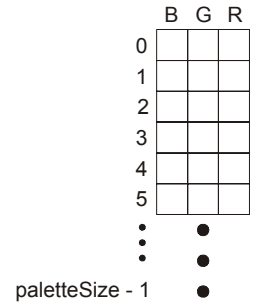


Ilustración 4-1 La clase *C_Image* incluye una paleta

Ejemplo

```
C_Image imagen ();
. . .
imagen.Palette (0, C_RED) = 255;
imagen.Palette (0, C_GREEN) = 0;
imagen.Palette (0, C_BLUE) = 0;
```

4.2.3 Constructores

Los constructores permiten crear un objeto de la clase. La clase *C_Image* dispone de varios constructores.

C_Image

C_Image ( )

Se usa para crear una imagen cuyo contenido vendrá dado por otra función. Por ejemplo, cuando se vaya a leer de un fichero.

C_Image

C_Image (const IndexT firstRow, const IndexT lastRow, const IndexT firstCol, const IndexT lastCol, const ElementT initValue = 0, const IndexT paletteSize = 256)

Se usa para crear una imagen asignándole un tamaño a la matriz de píxeles y, opcionalmente, a la paleta.

Parámetros de entrada

firstRow Índice de la primera fila.

lastRow Índice de la última fila. Ha de ser mayor o igual que *firstRow*.

firstCol Índice de la primera columna.

lastCol Índice de la última columna. Ha de ser mayor o igual que *firstCol*.

initValue Valor inicial para cada píxel. Por omisión, se inicializarán con 0.

paletteSize Tamaño de la paleta. O lo que es lo mismo, número de colores disponible en la imagen. La numeración comienza en 0 y termina en *paletteSize* - 1. Por omisión, se crea de 256 colores.

4.2.4 Destructores

Cuando un objeto *C_Image* ya no es necesario, se puede destruir, liberando así la memoria utilizada. Se emplea para eso el destructor de la clase.

~C_Image

~C_Image ( )

Destruye el objeto *C_Image*.

Free

Free ( )

Libera el espacio ocupado por la imagen, incluyendo la paleta, pero conserva el objeto. No es un destructor propiamente del lenguaje C++, pero se incluye porque en algunos casos puede resultar adecuado liberar sólo el cuerpo de la imagen por motivos de ahorro de memoria, pero mantener el objeto para usarlo posteriormente.

4.2.5 Acceso datos sobre la imagen

Además de los métodos heredados de la clase C_Matrix, por ejemplo para conocer el tamaño, se incluyen otros para acceder datos sobre la imagen.

PaletteSize

IndexT PaletteSize ( )

Devuelve el número de filas o colores de la paleta. El número de columnas es siempre 3.

SetPaletteSize

SetPaletteSize (const IndexT paletteSize)

Cambia el número de filas o colores de la paleta.

4.2.6 Otros métodos

Grey

Grey ( )

Convierte una imagen en color en una imagen en tonos de grises con 256 niveles. Modifica también la paleta de forma que albergue los 256 niveles de gris.

4.3 MÉTODOS DE ENTRADA Y SALIDA

ReadBMP

ReadBMP (const char * filePath)

Lee una imagen de un fichero en formato BMP. Este fichero ha de tener una imagen sin comprimir y almacenada con paleta.

Parámetros de entrada

filePath Nombre del fichero.

WriteBMP

WriteBMP (const char * filePath)

Escribe una imagen en un fichero en formato BMP con paleta.

Parámetros de entrada

filePath Nombre del fichero.

5 Preprocesamiento de Imágenes

J.A. PIEDRA

5.1 INTRODUCCIÓN

La fase del **preprocesado** está constituida por un conjunto de métodos que tienen la finalidad de realzar en la forma aquellas características que nos interesan para una aplicación concreta. La idea se basa en desechar todas aquellas características que no sean útiles para el proceso al que pretendamos someter a nuestra forma, al tiempo que se deben conservar aquellas que en alguna medida se consideren importantes.

Una definición más clara del preprocesado de una imagen es el procesamiento mediante un conjunto de técnicas de mejora de manera que resulte más adecuada la imagen obtenida que la original para un problema específico.

Se parte de una imagen original que es transformada en una o varias imágenes nuevas mediante una secuencia de operaciones aplicadas a la imagen original. Estas transformaciones, dependiendo de la imagen y del método o métodos que se apliquen, tendrán repercusión en facilitar o no la etapa tanto de segmentación como de extracción de características.

Este capítulo se centra en el uso de métodos en el dominio espacial. Los métodos del dominio espacial son los que actúan directamente sobre la imagen, es decir, la imagen por definición existe en el dominio espacial, donde se hace referencia al conjunto de píxeles que la componen. En contraposición, aparecen los métodos del dominio de las frecuencias que se basan en la transformación de la imagen para modificar los datos que la componen desde el dominio espacial a un nuevo dominio como es el de las frecuencias.

El desarrollo de métodos en el dominio espacial [1, 2, 3, 4, 5] y no en el de la frecuencia se centra principalmente en la optimización del computo de las operaciones. Es decir, trabajar en el dominio de la frecuencia implica desarrollar una serie de transformadas directas e inversas para modificar el contenido del dominio espacial y poder trasladarlo al de las frecuencias. El cálculo de la transformación para pasar de un dominio a otro implica un coste en el cálculo elevado, y si

además añadimos la operación en si a realizar esto se incrementa temporalmente aun más. Gráficamente se podría ver de la siguiente manera.

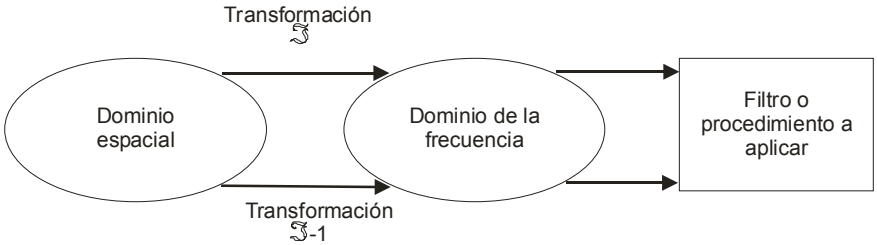


Figura 5–1 Gráfica del proceso del dominio de la frecuencia

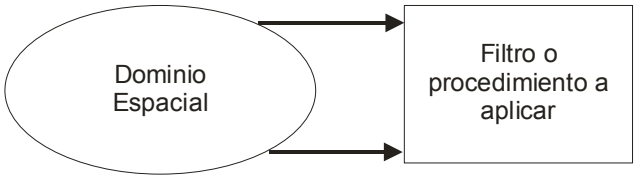


Figura 5–2 Gráfica del proceso del dominio espacial

Se observa que la forma más rápida de cálculo es mediante filtros en el dominio espacial. Por otra parte el dominio de la frecuencia no implica la obtención de una imagen mejor. Las sucesivas transformaciones a través de procedimientos en un computador cuyo dominio de trabajo es discreto y no real implica un acumulamiento del posible error que se este cometiendo.

Los métodos en el dominio espacial son procedimientos que operan directamente sobre los píxeles de una imagen. Dichas funciones pueden expresarse como

$$g(x, y) = T[f(x, y)]$$

Ecuación 5–1 Función de transformación

donde f(x, y) es la imagen de entrada, g(x, y) es la imagen procesada y T es un operador que actúa sobre f, definido en algún entorno de (x, y). La aproximación principal para definir un entorno alrededor de (x, y) es emplear un área de subimagen cuadrada o rectangular centrada en (x, y). El centro de la subimagen se mueve píxel a píxel comenzando, por ejemplo, en la esquina superior izquierda y aplicando el operador en cada posición (x, y) para obtener g.

La idea general consiste en determinar g en un punto (x, y) a partir de los valores de f en un entorno predefinido de (x, y). Una de las aproximaciones principales en este tipo de formulación se basa en el empleo de las denominadas **máscaras**. Básicamente, una máscara es una pequeña distribución bidimensional (por ejemplo, de 3×3), en la que los valores de los coeficientes determinan la naturaleza del proceso, como la acentuación de los bordes. Las

técnicas o mejoras basadas en este tipo de aproximación se conocen como **procesamiento por máscaras o filtrado**.

Durante el desarrollo de este capítulo trataremos de explicar el resultado de aplicar distintos tipos de filtros de imágenes basados en los métodos anteriores sobre una imagen para su mejora. Para poder explicar el resultado obtenido, aplicaremos cada uno de los filtros a dos imágenes que se muestran a continuación.

Figura 5–3 Imagen original 'aviongr.bmp'

Figura 5–4 Imagen original 'payaso.bmp'

Las características de estas imágenes son las siguientes:

- **'aviongr.bmp':** Imagen de 640 × 480 a 256 niveles de gris.
- **'payaso.bmp':** Imagen de 256 × 256 a 256 niveles de gris

Estas dos imágenes presentan características bien diferenciadas, en la primera los bordes están más difuminados y presenta mucho ruido, mientras que en la segunda, los bordes se

encuentran bien diferenciados y el ruido es casi inapreciable. Estas diferencias van a permitir una mejor comprensión acerca del funcionamiento de los distintos algoritmos.

5.2 MECANISMOS DE APROXIMACIÓN AL RANGO DE COLORES

En el cálculo de algunos filtros, así como la propia aplicación de la matriz de convolución, pueden generar que los valores de la imagen se salgan fuera del rango en el que han sido definidos.

Para controlar que el rango de los valores de la imagen no exceda los límites establecidos, se ha optado por realizar unos procedimientos incluidos en la librería 'TDI' dentro de la clase *C_Matrix*. Estos métodos permiten restringir un conjunto de valores de la matriz de la imagen y trasladarlos al rango de valores de los colores en el que verdaderamente se mueve la imagen.

Estos procedimientos suelen ser muy útiles a la hora de visualizar los distintos resultados que se han obtenido a través de distintos algoritmos de preprocesado o mejora de la imagen. De esta manera, se consigue controlar el valor de los distintos elementos de la matriz de la imagen, a la hora de ser almacenados en un archivo o mostrados por pantalla. Esto evita que se produzcan errores por parámetros que se encuentren fuera del rango de colores de la imagen.

5.2.1 Truncamiento

El primero de ellos, consiste en truncar directamente el valor del resultado o la imagen resultante en un intervalo definido. Gráficamente puede verse de la siguiente manera.

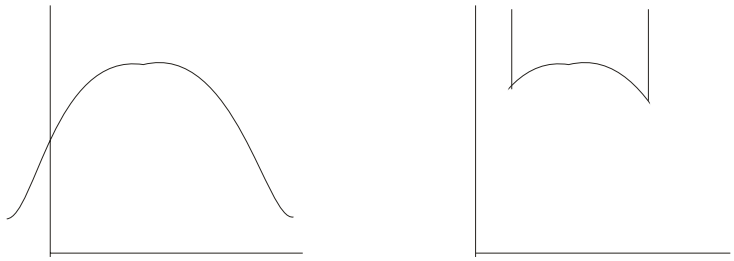


Figura 5–5 Gráfica de la función de una imagen normal y su truncada

El procedimiento se basa simplemente en aproximar un valor de la siguiente manera:

$$f(x) = \begin{cases} x & Si\ Min < x < Max \\ Min & Si\ x < Min \\ Max & Si\ Max < x \end{cases}$$

Ecuación 5–2 Función de truncado

Es decir, si el píxel excede los límites establecidos se aproxima por el máximo o mínimo dependiendo en su defecto del que más se aproxime.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Trunc (const ElementT min, const ElementT max).

  Calcula la matriz que hace referencia al método entre los valores de entrada, de manera que todo aquel elemento que exceda el rango de entrada se asigna uno de los valores.

- ***Parámetros de entrada***

  min valor mínimo del rango de entrada.

  max valor máximo del rango de entrada.

- ***Parámetros de salida***

  matriz de salida con los valores de los elementos dentro del rango de entrada.

Ejemplo

```
C_Image image,image2;
C_Matrix::IndexT rowN,colN,colorsN;
int isDark;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

image2 = image;
C_Trace("Crimmins Speckel Removing ...");
image.CrimminsFilter(image2,isDark);

C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
image.Trunc(0,(colorsN-1));

C_Trace ("Writting Filter Image...");
image.WriteBMP (OutFile);

image.Free();
image2.Free();
```

En este ejemplo actúa la función de truncamiento sobre una imagen a la que se le ha aplicado el filtro de *Crimmins*. El Min (mínimo de la paleta de la imagen) tomado para la función es 0 y el Max (máximo de la paleta de la imagen) es el número de colores de la paleta – 1.

5.2.2 Suavizado

El siguiente método es menos drástico que el anterior. Se basa en aplicar una función de suavizado a los distintos valores de la imagen. De esta manera, se consigue reducir el error que se obtiene al aplicar el truncamiento directamente, es decir, no hay un corte brusco en el rango de valores sino que cada uno de estos se aproxima al intervalo. De manera grafica podría verse como:

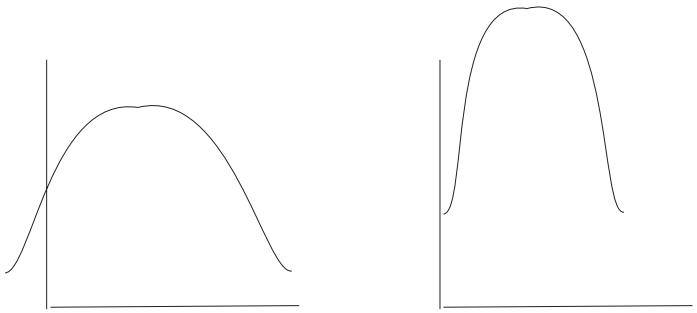


Figura 5–6 Gráfica de la función de una imagen normal y su función de suavizado

La función que define este comportamiento es la siguiente:

$$f(x) = \left(\frac{(Max - Min)}{(\max - \min)} \times x \right) + \min$$

Ecuación 5–3 Función de suavizado

donde Max y Min son los extremos del intervalo de colores definidos para la imagen, mientras que max y min son los valores máximo y mínimo alcanzados en la imagen al aplicar sobre ella una función de transformación.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Stretch (const ElementT minIn, const ElementT maxIn).

Calcula la matriz que hace referencia al método entre los valores de entrada. Aplica una función de suavizado llevando cada uno de los elementos de la matriz a un rango de entrada definido.

- ***Parámetros de entrada***

  minIn valor mínimo del rango de entrada.

  maxIn valor máximo del rango de entrada.

- ***Parámetros de salida***

  matriz de salida con los valores de los elementos dentro del rango de entrada.

Ejemplo

```
C_Image image,image2;
C_Matrix::IndexT rowN,colN,colorsN;
int isDark;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

image2 = image;
C_Trace("Crimmins Speckel Removing ...");
image.CrimminsFilter(image2,isDark);

C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
image.Stretch(0,(colorsN-1));

C_Trace ("Writting Filter Image...");
image.WriteBMP (OutFile);

image.Free();
image2.Free();
```

En este ejemplo actúa la función de suavizado sobre una imagen a la que se le ha aplicado el filtro de *Crimmins*. El Min (mínimo de la paleta de la imagen) tomado para la función es 0 y el Max (máximo de la paleta de la imagen) es el número de colores de la paleta – 1 para llevar a cabo la sucesión de 0 a dicho color.

5.3 MÉTODOS DE PROCESAMIENTO DE PUNTOS

5.3.1 Función Inversa

La aplicación de la función [2] inversa es de gran utilidad para algunas aplicaciones. Entre estas aplicaciones cabe destacar la representación de imágenes médicas, la obtención de

fotografías de una pantalla con película monocroma con la idea de emplear los negativos resultantes como diapositivas normales.

La función inversa de una imagen se obtiene al aplicar un método sobre los niveles de gris de la imagen. La función de transformación s = T(r) donde s va a ser el resultado de aplicar la inversa modulo L al valor de r, y donde L es el número de niveles de gris. La idea es invertir el orden de blanco a negro, de forma que la intensidad de la imagen de salida disminuya conforme la intensidad de la imagen de entrada aumente. La función que aplica T sobre la imagen original se denomina función inversa.

La gráfica siguiente sobre un histograma cualquiera puede darnos una idea de la forma de la función.

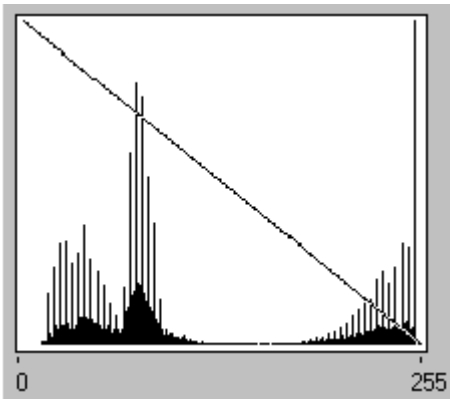


Figura 5–7 Función Inversa para un histograma cualquiera

El procedimiento que implementa la función inversa es bastante sencillo. Simplemente, el resultado almacenado en cada píxel va a ser el de restar a un escalar el valor del píxel de la imagen fuente. Normalmente el escalar a tomar si es para una imagen en niveles de gris es el 255 que coincide con el valor del color blanco en la paleta.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Negative (const ElementT escalar).

  Donde escalar contiene el valor a restar sobre cada uno de los elementos que componen la matriz de la imagen.

- ***Parámetros de entrada***

  escalar del tipo ElementT definido.

- ***Parámetros de salida***

  matriz de salida.

Ejemplo

```
C_Image image;
C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);

C_Trace("Calculating Negative image");
image.Negative((colorsN-1));

C_Trace ("Writting Negative Image...");
image.WriteBMP (OutFile);

image.Free();
```

En este ejemplo se muestra como se aplica la función inversa a una imagen para obtener el negativo de la imagen. El valor escalar a restar es el número de colores de la imagen menos 1 ya que la sucesión se inicia en 0. Para los resultados que se muestran a continuación el valor del escalar es 256 –1.

Resultados

El resultado de aplicar este filtro a una imagen en niveles de gris, es la misma imagen pero con los niveles de gris cambiados. Es decir, aquellos píxeles cuyo color original sea 0 en la imagen de salida presentaran un valor 255, un píxel original con valor 1 presentara 254 en la salida, y así con todos los píxeles de la imagen.

Estos son los resultados obtenidos para las dos imágenes originales:

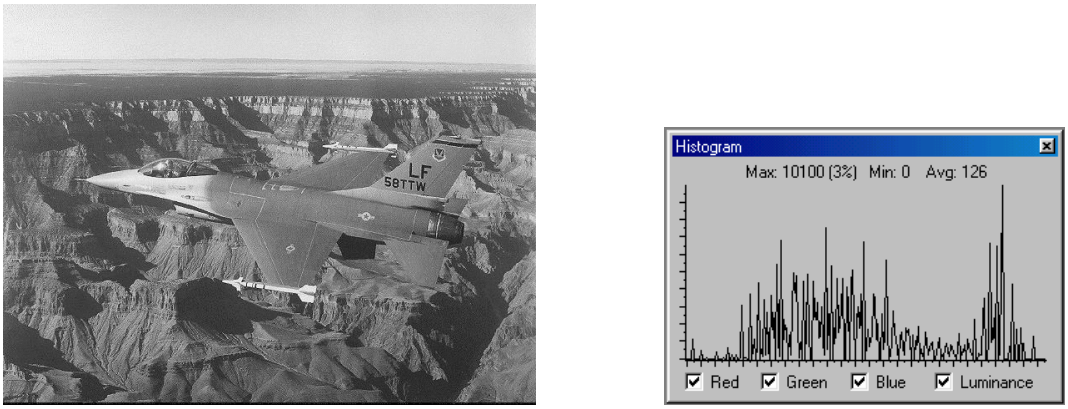


Figura 5–8 Imagen original ‘aviongr.bmp’ y su histograma

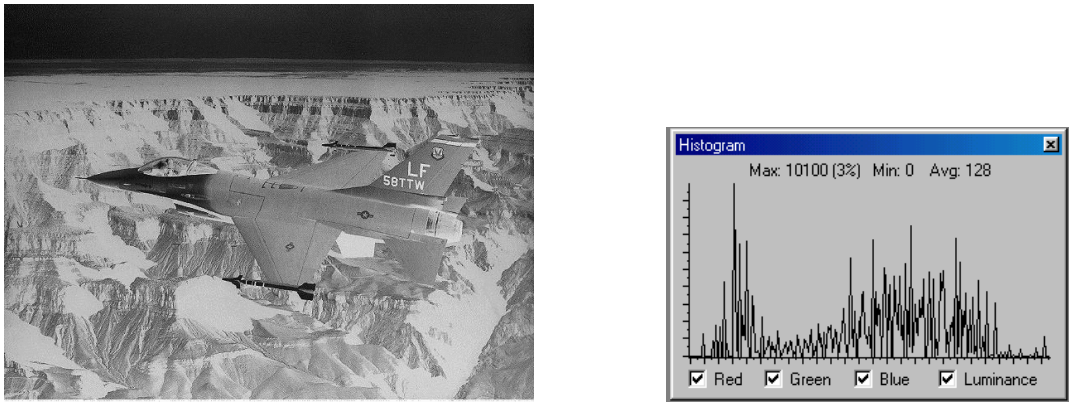


Figura 5–9 Imagen inversa ‘aviongr.bmp’ y su histograma

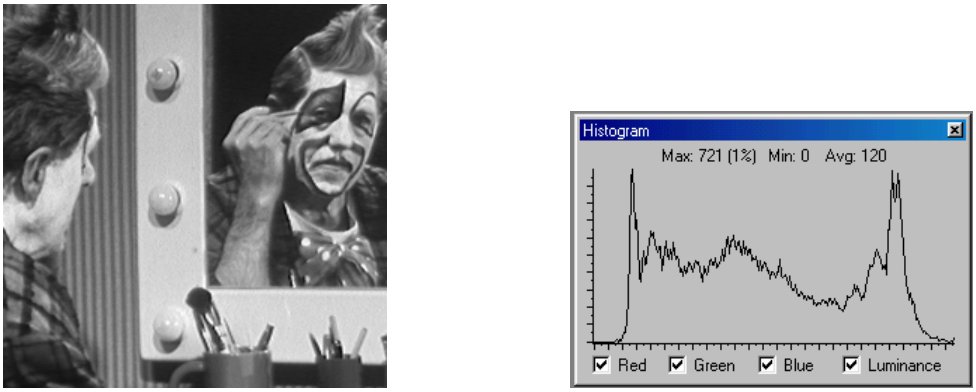


Figura 5–10 Imagen original ‘payaso.bmp’ y su histograma

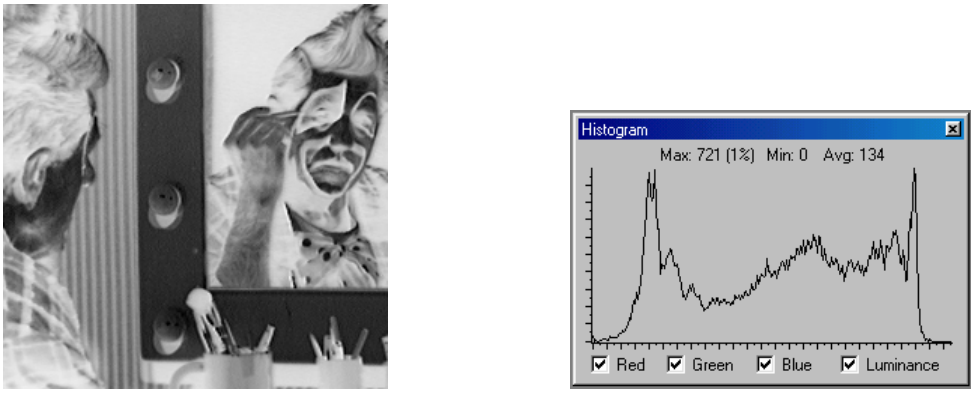


Figura 5–11 Imagen inversa ‘payaso.bmp’ y su histograma

Las imágenes resultantes obtenidas como se pueden comprobar son inversas a las originales, es decir, en las zonas donde los colores son más claros aparecen colores más oscuros y en las zonas donde los colores son más oscuros aparecen colores más claros. Esto es más fácilmente apreciable en cada uno de los histogramas que se muestran, donde se observa que las gráficas tanto del histograma de la imagen original como el de la imagen inversa si se unieran serían simétricas, es decir, la forma de los histogramas se encuentra invertida de derecha a izquierda.

5.3.2 Aumento del contraste

La carencia de contraste [2, 4]en las imágenes puede deberse a multitud de factores, como iluminación deficiente, falta de rango dinámico en el sensor o incluso incorrecta selección de la apertura de la lente durante la captación de la imagen. La idea en las técnicas de **aumento del contraste** es aumentar el rango dinámico de los niveles de gris de la imagen.

Una transformación típica empleada para la mejora del contraste es la representada en la Figura 5–12. La ubicación de los puntos (r_1, s_1) y (r_2, s_2) controla la forma de la función de transformación. Por ejemplo, si $r_1 = s_1$ y $r_2 = s_2$, la transformación es una función lineal que no produce cambios en los niveles de gris. Si $r_1 = r_2$, $s_1 = 0$ y $s_2 = L-1$, la transformación se convierte en una función umbral que crea una imagen binaria. A continuación, se muestra gráficamente cual sería la función de aumento de contraste.

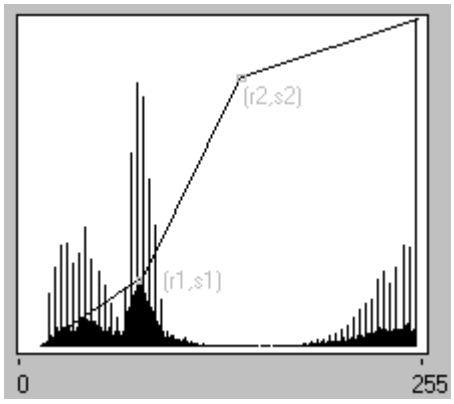


Figura 5–12 Función de aumento de contraste para un histograma cualquiera

El procedimiento que realiza un aumento de contraste sobre una imagen se basa en el cálculo de las distintas líneas que se aprecian en la figura a través de la ecuación punto pendiente. Para las rectas las ecuaciones son las siguientes:

- Ecuación formada por los puntos (mínimo, mínimo) y (r_1, s_1):

$$y = \left(\left(\frac{(s_1 - \min)}{(r_1 - \min)} \right) \times (x - \min) \right) + \min$$

Ecuación 5–4 Fórmula de la recta inicial

- Ecuación formada por los puntos (r_1, s_1) y (r_2, s_2):

$$y = \left(\left(\frac{(s_2 - s_1)}{(r_2 - r_1)} \right) \times (x - r_1) \right) + s_1$$

Ecuación 5–5 Fórmula de la recta intermedia

- Ecuación formada por los puntos (r_2, s_2) y (máximo, máximo):

$$y = \left(\left(\frac{(\max - s_2)}{(\max - r_2)} \right) \times (x - r_2) \right) + s_2$$

Ecuación 5–6 Fórmula de la recta final

El módulo recibe como parámetros los valores de los puntos (r_1, s_1) y (r_2, s_2) que definen el rango de la imagen donde se va a producir el aumento de contraste. Recorre cada píxel para ajustar ese rango a los valores contenidos en la imagen.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Contrast (const ElementT r1,const ElementT s1,const ElementT r2,const ElementT s2,const ElementT min, const ElementT max).

  Cada uno de los parámetros de entrada hace referencia a los valores explicados en las ecuaciones anteriores. Se limita a llevar el rango de valores de la matriz al rango especificado.

- ***Parámetros de entrada***

  r1 valor de la imagen original (nivel de gris).

  s1 valor deseado de la imagen original (nivel de gris).

  r2 valor de la imagen original (nivel de gris).

  s2 valor deseado de la imagen original (nivel de gris).

  min valor mínimo posible de la imagen original (se puede o no corresponder con el valor mínimo alcanzado en la imagen).

  max valor máximo posible de la imagen original (se puede o no corresponder con el valor máximo alcanzado en la imagen).

- ***Parámetros de salida***

  matriz de salida.

Ejemplo

```
C_Matrix::ElementT r1=130,s1=130,r2=130,s2=130;
C_Image image;
C_Matrix::IndexT rowN,colN,colorsN;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
```

```
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Calculating Contrast Image");
// r1=60 | s1=100 | r2=180 | s2=220 .
image.Contrast(r1,s1,r2,s2,0,(colorsN-1));

C_Trace ("Writting Contrast Image...");
image.WriteBMP (OutFile);

image.Free();
```

Resultados

La finalidad versa en poder llevar el rango de colores a los niveles deseados para trabajar mejor con la imagen en cuestión. Se consigue a través de un análisis previo del histograma de la propia imagen.

Para mostrar los resultados obtenidos vamos a utilizar el siguiente rango de valores: (r_1, s_1) = (60, 100) y (r_2, s_2) = (180, 220) para aplicarlo sobre cada una de las distintas imágenes. Se han obtenido las imágenes representadas en la siguiente figura:

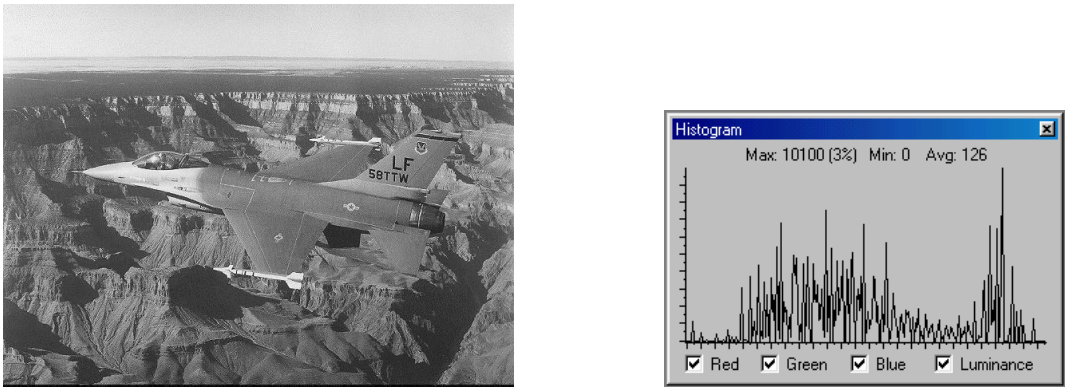


Figura 5–13 Imagen original ‘aviongr.bmp’ y su histograma

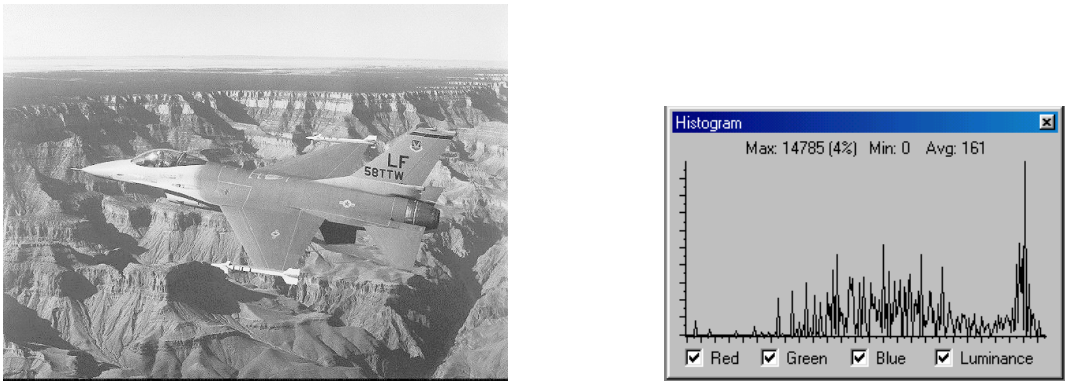


Figura 5–14 Imagen con aumento de contraste ‘aviongr.bmp’ y su histograma. (r1=60, s1=100) - (r2=180, s2=220)

Para la Figura 5–14, el contraste aplicado permite esclarecer la imagen. Se puede observar que realiza una especie de umbralizado pero suavizando el mismo, de esta manera queda

delimitado un umbral más suave del histograma. Se aprecian perfectamente el mínimo y el máximo establecidos al que se les aplica el contraste.

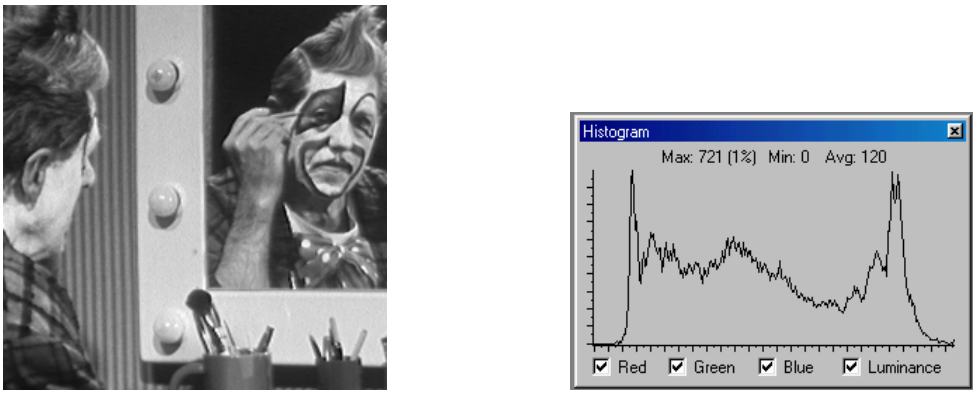


Figura 5–15 Imagen original ‘payaso.bmp’ y su histograma

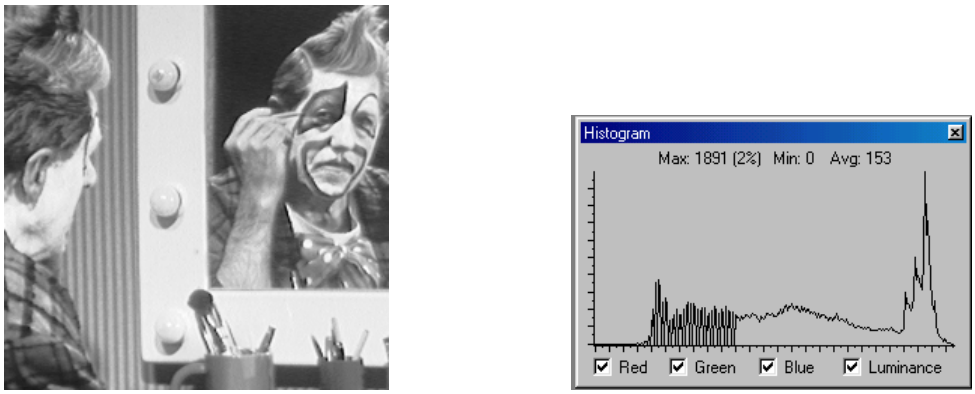


Figura 5–16 Imagen con aumento de contraste ‘payaso.bmp’ y su histograma. (r1=60, s1=100) - (r2=180, s2=220)

A partir de la Figura 5–16 anterior, al igual que sucedía con la Figura 5–14, podemos ver el aumento de la luminosidad de la imagen como producto de unos parámetros de contraste alto, provocan un esclarecimiento de la imagen original.

Suele ser útil cuando es necesario tomar imágenes donde el contraste de los objetos a priori es muy similar y el nivel de luminosidad no es el adecuado. El aplicar el aumento de contraste implica una mejor diferenciación de los distintos objetos que componen una imagen.

5.3.3 Ecualización del histograma

Previamente a la ecualización del histograma [2, 3]es necesario definir el histograma en sí. El histograma de una imagen digital con niveles de gris en el rango [0, L-1] es una función discreta

$$p(rk) = nk / n$$

Ecuación 5–7 Función del histograma

donde r_k es el k-ésimo nivel de gris, n_k es el número de píxeles de la imagen con ese nivel de gris, n es el número total de píxeles de la imagen y $k = 0, 1, 2, ..., L-1$.

La función del histograma $p(r_k)$ refleja el valor de la probabilidad en el nivel de gris r_k. La representación gráfica de esta función para todos los valores de k proporciona una descripción global de la apariencia de una imagen. Por lo tanto, el histograma permite establecer si una imagen tiene una apariencia oscura, si el rango dinámico de valores es estrecho, si una imagen tiene un bajo o alto contraste, etc. El perfil del histograma de una imagen proporciona información muy útil sobre la posibilidad de mejora de la imagen.

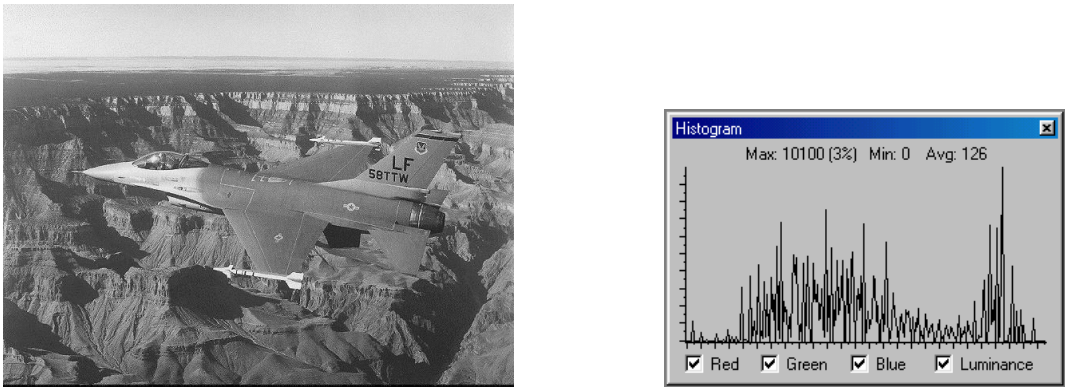


Figura 5–17 Histograma de la imagen 'aviongr.bmp'

Si antes de visualizar la imagen original, tuviésemos que sacar conclusiones globales sobre la imagen con tan solo el histograma de la imagen, podemos decir que la imagen presenta dos rangos de colores que son los mas utilizados de los *256* colores en escala de gris. Un conjunto de colores oscuros que corresponden al fondo de la imagen, es decir, las montañas que aparecen y un conjunto de colores claros, casi blancos, que corresponden al avión contenido y al cielo en el que vuela.

La **ecualización del histograma** se puede definir de la siguiente manera:

Sea r una variable que represente los niveles de gris de la imagen a mejorar. En la parte inicial de la presentación se supondrá que los píxeles son cantidades continuas que han sido normalizadas de forma que pertenezcan al intervalo $[0, 1]$, con $r = 0$ representado el negro y $r = 1$ el blanco. Posteriormente consideraremos una formulación discreta y permitiremos que los valores del píxel varíen en el intervalo $[0, L - 1]$.

Para cada r del intervalo $[0, 1]$, nos centramos en las transformaciones de la forma

$$s = T(r)$$

Ecuación 5–8 Operador de transformación

Que producen un nivel *s* para cada valor de píxel *r* de la imagen original. Se supone que el valor de *s* verifica las condiciones:

a) *T(r)* es de valor único y monótonamente creciente en el intervalo $0 \le r \le 1$; y

b) $0 \le T(r) \le 1$ para $0 \le r \le 1$.

La condición *a)* preserva el orden entre blanco y negro de la escala de grises, mientras que la condición *b)* garantiza una aplicación que es coherente con el rango de valores de píxel permitidos.

Para poder ser de aplicación en el procesado digital de imágenes, los conceptos que se acaban de desarrollar han de expresarse en forma discreta. Para los niveles de gris que constituyen los valores discretos, se tienen las probabilidades:

$$Pr(rk) = nk / n \; 0 \le rk \le 1 \text{ y } k = 0, 1, ..., L - 1$$

Ecuación 5–9 Cálculo de probabilidades

donde, *L* es el número de niveles, $p_r(s_k)$ es la probabilidad del *k*-esimo nivel de gris, n_k es el número de veces que este nivel aparece en la imagen y *n* es el número total de píxeles de la imagen. Una representación gráfica de $p_r(r_k)$ en función de r_k se denomina un histograma y la técnica empleada para obtener un histograma uniforme se conoce como ecualización del histograma o linealización del histograma.

La forma discreta de la ecuación utilizada para el calculo de la ecualización del histograma es:

$$s_k = T(r_k) = \sum j / n = \sum p_r(r_j) \qquad 0 \le rk \le 1 \text{ y } k = 0, 1, ..., L - 1$$

Ecuación 5–10 Ecualización del histograma

La transformación inversa se indica por

$$r_k = T^{-1}(s_k) \qquad 0 \le sk \le 1$$

Ecuación 5–11 Inversa de la ecualización del histograma

donde tanto $T(r_k)$ como $T^{-1}(s_k)$ se supone que verifican las condiciones *a)* y *b)*. La función de transformación $T(r_k)$ puede calcularse directamente de la imagen a partir de las ecuaciones anteriores. La función $T^{-1}(s_k)$ no se emplea en ecualización de los histogramas.

El primer paso para obtener la ecualización del histograma de una imagen es calcular el histograma de la imagen. Esta tarea se realiza mediante el siguiente procedimiento.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  CalculateHistogram(C_Matrix &histogram).

  Calcula el histograma a partir de la matriz de la imagen.

- ***Parámetros de entrada***

  histogram matriz del histograma.

- ***Parámetros de salida***

  histogram matriz del histograma calculada.

El histograma para una imagen en niveles de gris es una matriz de [0*, 255*]. En cada posición de la matriz se almacena el número de píxeles que presentan ese valor en la imagen. La ecualización del histograma se realiza de la siguiente manera:

La interfaz de acceso al procedimiento

- ***Procedimiento***

  EqualityHistogram(C_Matrix &hist).

  Calcula el histograma ecualizado a partir del histograma de la matriz de la imagen. Como resultado aplica está ecualización a la imagen.

- ***Parámetros de entrada***

  hist matriz del histograma.

- ***Parámetros de salida***

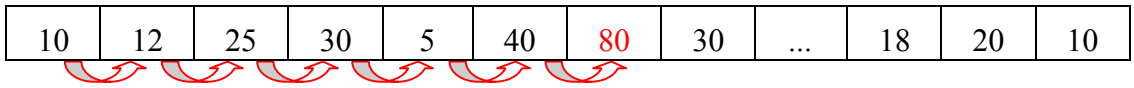
  matriz de salida con la aplicación del histograma ecualizado.

Este procedimiento se puede descomponer en tres etapas.

1. **Cálculo del acumulador.**

   Se trata de calcular el valor acumulado para una posición del histograma con respecto a sus antecesores, es decir, la suma del valor contenido en la posición y de cada uno de los valores contenidos en sus antecesores.

   Ejemplo: Cálculo del valor acumulado para la posición 6 del histograma que corresponde con el color nº 6 de la paleta.



*Valor acumulado de la posición 6 = 10+12+25+30+5+40+80 =**202***

2. **Obtención de la frecuencia acumulativa.**

   Posteriormente, para cada píxel de la imagen se obtiene el valor de la frecuencia como el valor de cada posición del histograma entre el tamaño de la imagen.

   Ejemplo: Cálculo de la frecuencia acumulativa para un píxel de la imagen cuyo valor es 6.(Tamaño de la imagen 40 × 50).

   *Valor de la frecuencia acumulativa de la posición 6 = 202 / 2000 = **0.101***

3. **Asignación del valor al píxel de la imagen.**

   Para cada píxel se asigna el máximo de 0 y el valor de la frecuencia multiplicado por 256 menos 1.

   Ejemplo: Asignación para un píxel de la imagen cuyo valor es 6.(Tamaño de la imagen 40 × 50).

   *Valor de asignación al píxel con valor 6 = Máximo(0, 0.101*255) = **25.755***

Cabe destacar que los niveles de gris de una imagen que ha sido sometida a la ecualización del histograma están repartidos y siempre alcanzan el blanco. Este proceso incrementa el rango dinámico de niveles de gris y, en consecuencia, produce un aumento en el contraste de la imagen. Para presentar el resultado obtenido en la ecualización presentaremos la imagen original junto con su histograma y la imagen resultante de la ecualización junto con su histograma.

Ejemplo

```
C_Image image;
C_Matrix::IndexT rowN,colN,colorsN;

C_Trace ("Reading image");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);

C_Matrix histogram(1,1,0,(colorsN-1),0);
C_Trace ("Calculating Histogram..");
image.CalculateHistogram(histogram);
```

```
C_Trace ("Writting Histogram...");
histogram.Write("Histograma.txt");

C_Trace ("Equality of Histogram...");
image.EqualityHistogram(histogram);
```

En este ejemplo se muestra el cálculo del histograma, así como la ecualización del histograma para una imagen dada.

Resultados

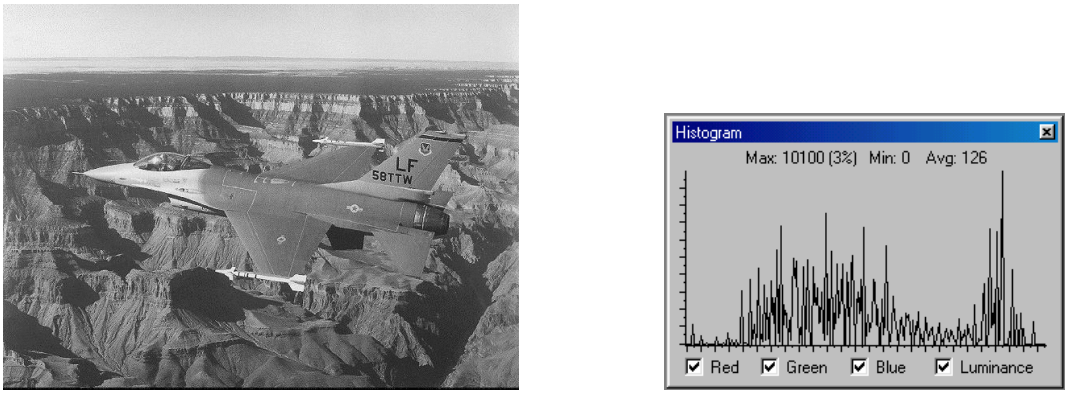


Figura 5–18 Imagen original ‘aviongr.bmp’ y su histograma

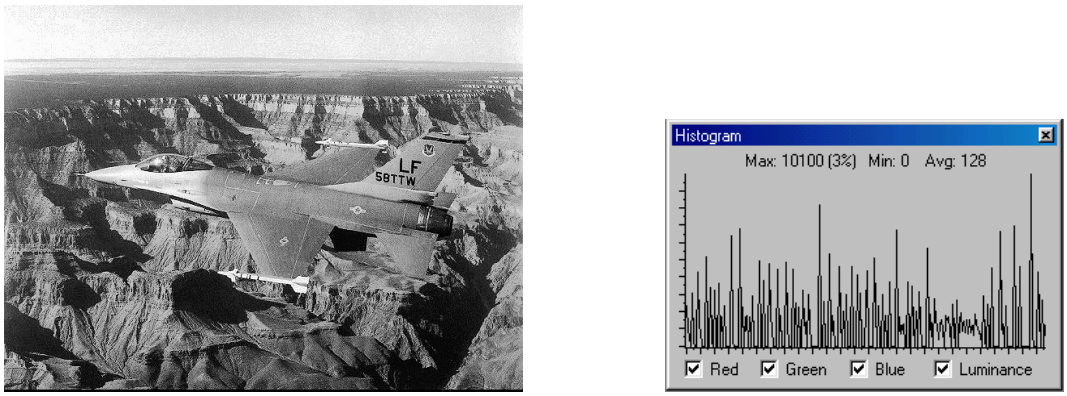


Figura 5–19 Imagen ecualizada ‘aviongr.bmp’ y su histograma uniforme

En las imágenes anteriores podemos ver como los niveles de gris utilizados en la imagen se distribuyen de manera uniforme. La mejora respecto a la imagen original resulta evidente.

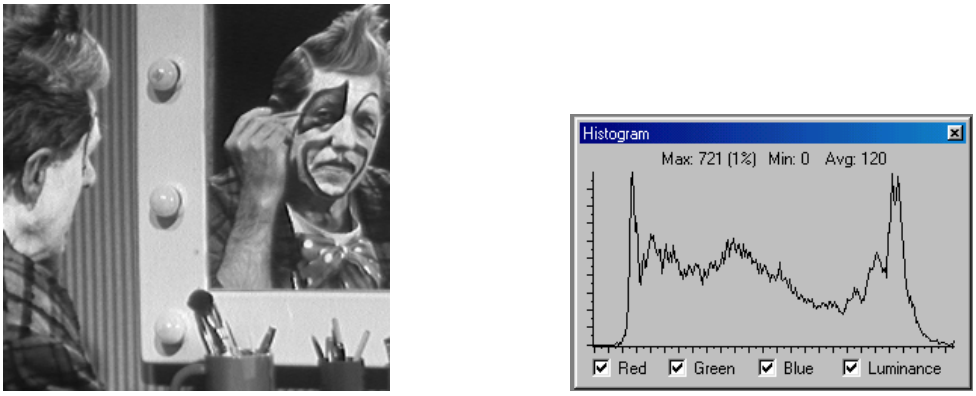


Figura 5–20 Imagen original 'payaso.bmp' y su histograma

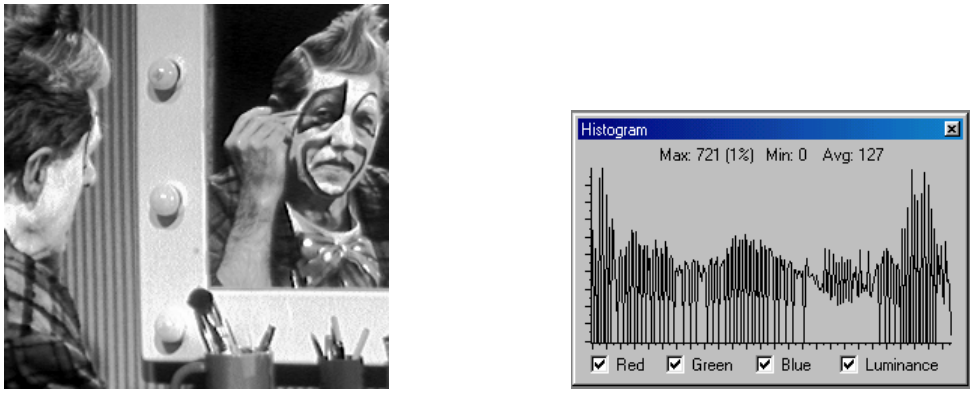


Figura 5–21 Imagen ecualizada 'payaso.bmp' y su histograma uniforme

En esta segunda imagen podemos apreciar mejor como la ecualización del histograma presenta una distribución de los niveles de píxeles de una imagen entre el rango de valores de niveles de gris. En la primera de las imágenes los niveles de gris utilizados estaban distribuidos sobre todo el rango de valores. En esta segunda imagen el rango de valores utilizados estaba más concentrado. La ecualización del histograma produce una dispersión de los niveles de gris llegando a utilizar los 256 colores.

5.3.4 Filtro exponencial

La **función exponencial** es un operador sin forma [5] que puede aplicarse a las imágenes de escala de grises. Se utiliza para cambiar el rango dinámico de una imagen. Esta función incrementa la intensidad de los valores de los píxeles.

El operador exponencial es un proceso a nivel de píxel donde la función de muestreo es una curva exponencial. Esto significa que el valor de la intensidad de cada píxel en la imagen de salida es igual que un valor elevado al valor del píxel correspondiente en la imagen de entrada. Cada número de base se utiliza dependiendo del grado de comprensión del rango dinámico. Para elevar la vista de una fotografía, valores por encima de uno son bastantes útiles. Para presentar el

resultado, la imagen debe ser escalada de tal forma que el valor máximo tiene que ser 255. la imagen resultante viene dada por la siguiente función:

$$Q(i,j) = cb^{P(i,j)}$$

Ecuación 5–12 Función exponencial

donde P y Q son las imágenes de entrada y salida, respectivamente, b es un valor base y c es el factor de escala. Para evitar resultados fuera de rango, muchas implementaciones restan 1 al termino exponencial antes de escalar. De esta forma se obtiene la siguiente formula:

$$Q(i,j) = c\left(b^{P(i,j)} - 1\right)$$

Ecuación 5–13 Función exponencial mejorada

La siguiente figura muestra la función para b = 10 y c = 1.01

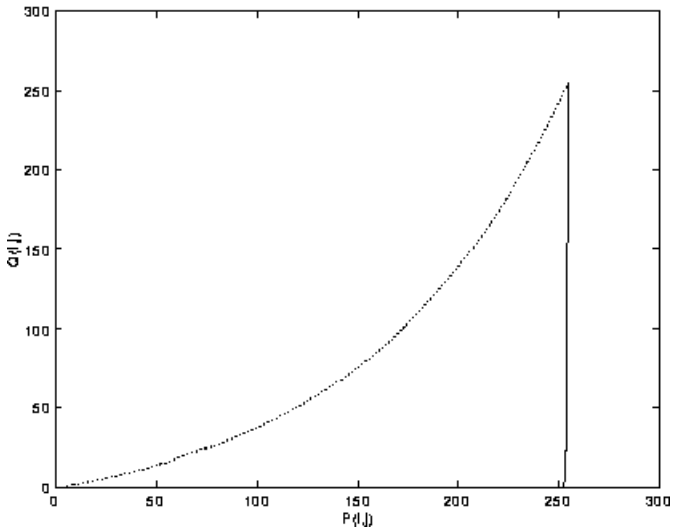


Figura 5–22 Función exponencial

Podemos ver en la figura que la curvatura de la función exponencial en base 10 esta muy cercano a incrementar la visión de una imagen normal. Podemos controlar la curvatura de la función escogiendo valores apropiados para los valores base.

A continuación se muestra la función aplicada a un histograma con niveles de gris.

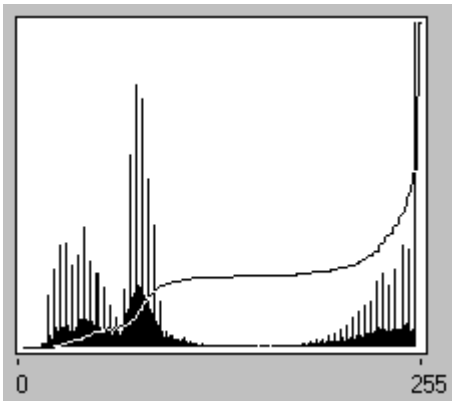


Figura 5–23 Función exponencial aplicada a un histograma

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Exponential(const ElementT b, const ElementT min, const ElementT max).
  Calcula la exponencial de la matriz de una imagen a partir de los parámetros de entrada como es la base de la ecuación anterior, el mínimo y máximo que puede alcanzar la imagen.

- ***Parámetros de entrada***

  b base de la exponencial.

  min nivel mínimo de gris que puede alcanzar la imagen (el color más bajo para los niveles de gris es el 0).

  max nivel máximo de gris que puede alcanzar la imagen (el color más alto para los niveles de gris es el 255).

- ***Parámetros de salida***

  matriz de salida con la aplicación de la exponencial.

Ejemplo

```
C_Matrix::ElementT c,b;
C_Matrix::IndexT rowN,colN,colorsN;
C_Image image;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);

C_Trace("Calculating Exponential Image");
image.Exponential(b,0,(colorsN-1));
```

```
C_Trace ("Writting Exponential Image...");
image.WriteBMP (OutFile);

image.Free();
```

Resultados

Los resultados obtenidos para cada una de las imágenes son los que se muestran a continuación.

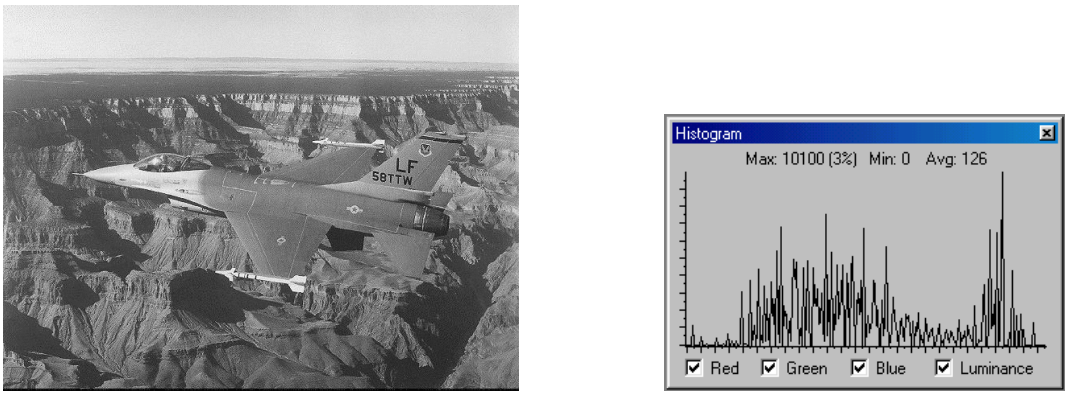


Figura 5–24 Imagen original ‘aviongr.bmp’ y su histograma

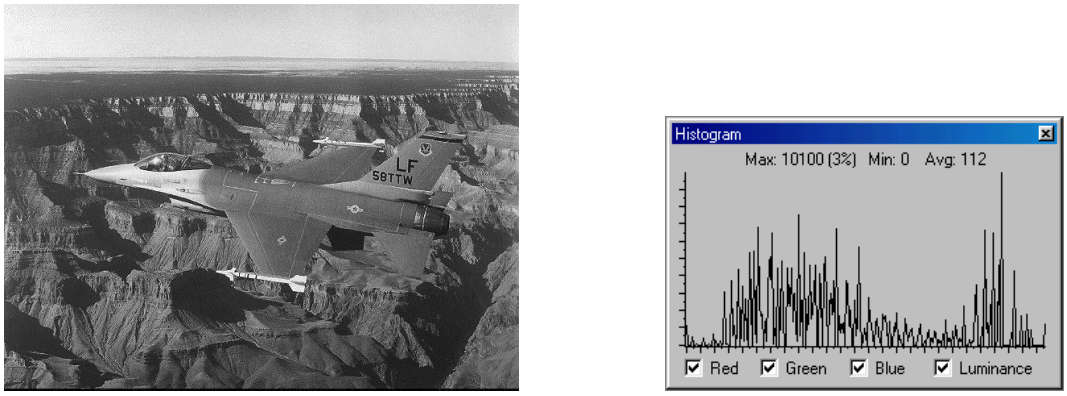


Figura 5–25 Imagen exponencial de base = 2 ‘aviongr.bmp’ y su histograma

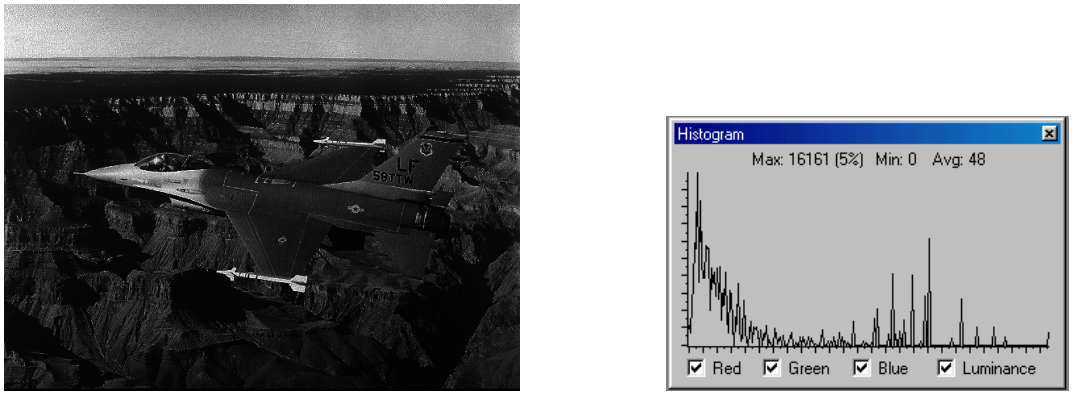


Figura 5–26 Imagen exponencial de base = 100 ‘aviongr.bmp’ y su histograma

Se observa, a medida que se incrementa la base se oscurece más la imagen, esto se debe no solo a la función exponencial sino al efecto que causa la función de suavizado sobre la imagen obtenida llevando los valores muy próximos al mínimo. El histograma obtenido bajo estos factores es inverso al de la función exponencial.

Para la otra imagen los resultados obtenidos son los siguientes:

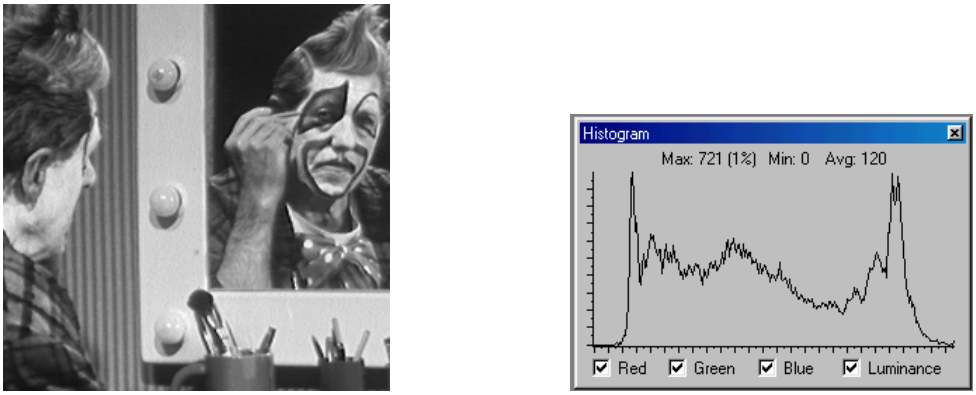


Figura 5–27 Imagen original 'payaso.bmp' y su histograma

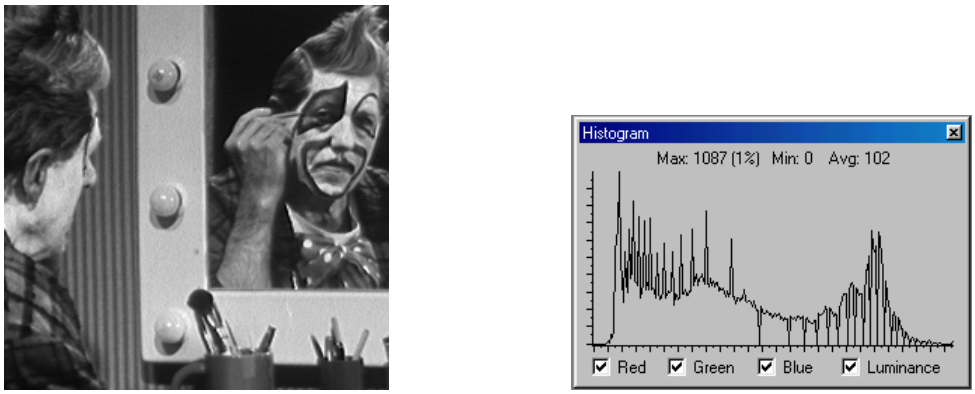


Figura 5–28 Imagen exponencial de base = 2 'payaso.bmp' y su histograma

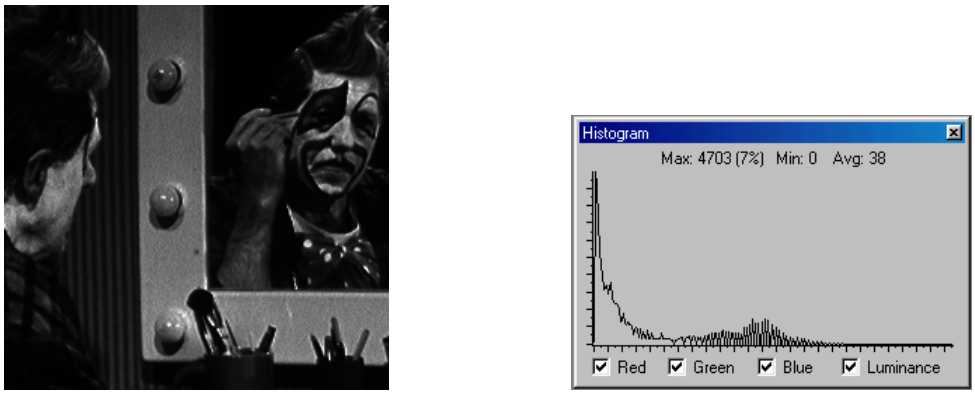


Figura 5–29 Imagen exponencial de base = 100 'payaso.bmp' y su histograma

En ambos resultados de las imágenes se observa como a medida que se incrementa el valor de b el histograma se aproxima más a la función exponencial. De esta manera se contempla un

oscurecimiento parcial de la imagen. El filtro exponencial oscurece las imágenes dándole más prioridad a los niveles de gris más bajos. En nuestro ejemplo hemos utilizado b = 2 y b = 100, donde valores más altos de b producirían un incremento de la intensidad de los niveles de gris más bajos de la imagen original.

5.3.5 Filtro logarítmico

El rango dinámico de una imagen [5] puede comprimirse reemplazando cada valor de píxel por su logaritmo. Esto produce el efecto, que los valores de los píxel de baja intensidad son intensificados. Aplicando el operador logaritmo a una imagen puede ser beneficioso en aplicaciones donde el rango dinámico puede ser demasiado largo para presentarlo en una pantalla.

El operador logarítmico es un procesamiento del píxel simple donde la función de muestreo es una curva logarítmica. Es decir, cada píxel es reemplazado con su logaritmo. La mayoría de las implementaciones aplican el logaritmo natural o el logaritmo en base 10.

La función de muestreo del logaritmo viene dada por

$$Q(i,j) = c\log\left(\left|P(i,j)\right|\right)$$

Ecuación 5–14 Función logarítmica

donde P y Q son las imágenes de entrada y salida, respectivamente, y c es el factor de escala. Dado que el logaritmo no esta definido para el valor 0, muchas implementaciones de este operador añaden el valor 1 a la imagen antes de aplicar el logaritmo. Este operador se define como

$$Q(i,j) = c\log\left(1+\left|P(i,j)\right|\right)$$

Ecuación 5–15 Función logarítmica mejorada

El valor de la constante c se elige de tal forma que el valor máximo de la salida sea 255. Eso ocurre si R es el valor con una magnitud máxima en una imagen de entrada, c viene dado por

$$c = \frac{255}{\log\left(1+\left|R\right|\right)}$$

Ecuación 5–16 Obtención de la constante c

El grado de compresión puede controlarse ajustando el rango de los valores de entrada. Dado que la función logarítmica llega a ser más lineal que la original, la compresión es más

pequeña para una imagen que contiene valores pequeños. La función de muestreo se representa para dos diferentes rangos de entrada en la siguiente figura:

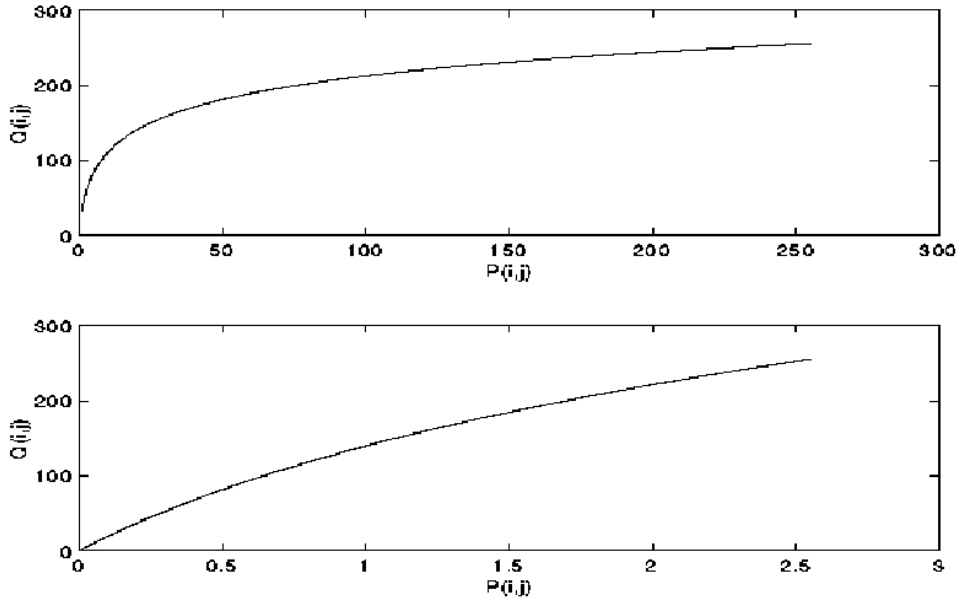


Figura 5–30 Distribución de la función logarítmica

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Logarithm(const ElementT min, const ElementT max).

  Calcula el logaritmo a la matriz de la imagen a partir los valores de máximo y mínimo que puede tomar la imagen.

- ***Parámetros de entrada***

  min mínimo posible a tomar por la imagen.

  max máximo posible a tomar por la imagen.

- ***Parámetros de salida***

  matriz de la imagen de salida con el logaritmo aplicado.

Ejemplo

```
C_Image image;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
```

```
C_Trace("Calculating Logarithm Image");
image.Logarithm(0,(colorsN-1));

C_Trace ("Writting Logarithm Image...");
image.WriteBMP (OutFile);

image.Free();
```

Resultados

Los resultados obtenidos para las distintas imágenes son los siguientes:

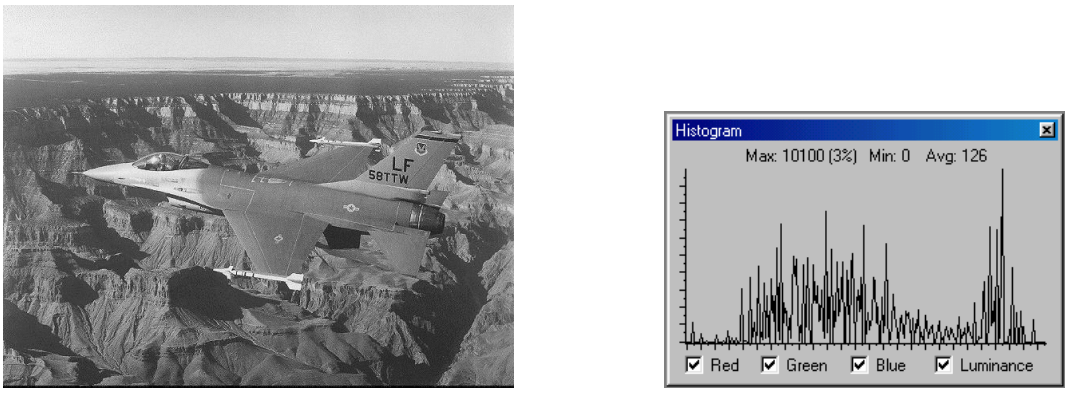


Figura 5–31 Imagen original ‘aviongr.bmp’ y su histograma

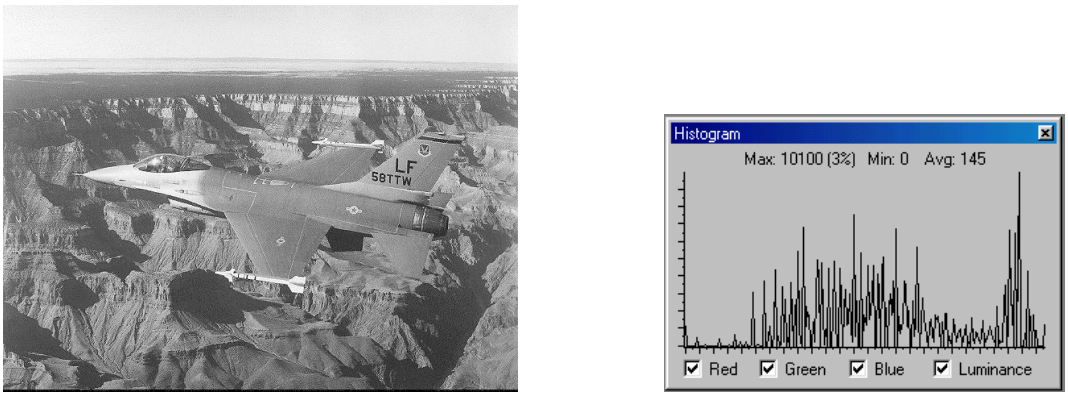


Figura 5–32 Imagen logarítmica ‘aviongr.bmp’ y su histograma

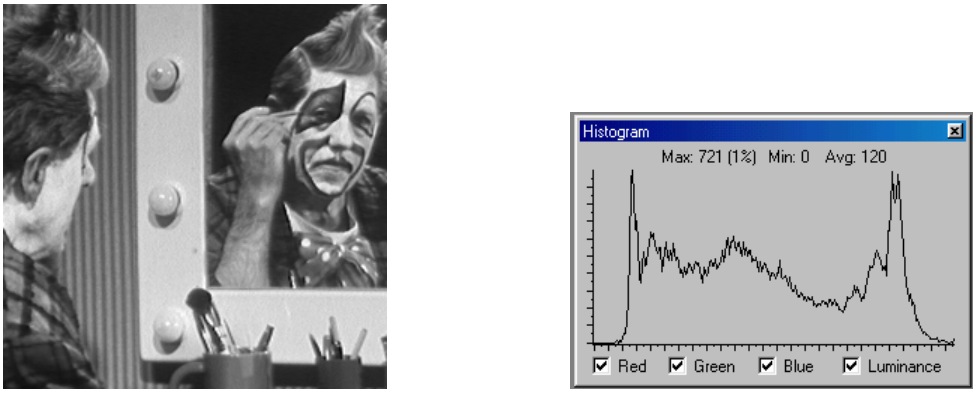


Figura 5–33 Imagen original ‘payaso.bmp’ y su histograma

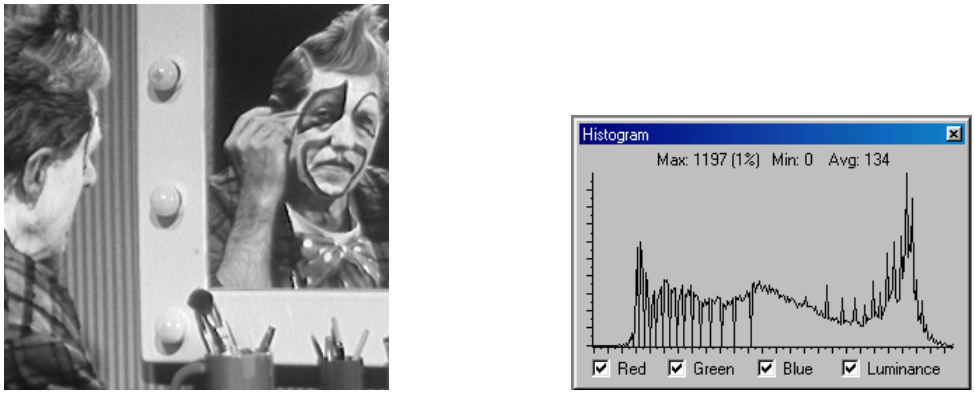


Figura 5–34 Imagen logarítmica 'payaso.bmp' y su histograma

Se observa que para los resultados obtenidos el logaritmo incrementa la intensidad en los colores más claros de la imagen. Atendiendo a la gráfica que muestra el histograma para cada una de las imágenes, se ve perfectamente el aumento de los niveles de gris más claros.

5.3.6 Función Gamma

En un monitor o en un televisor, [5,6] la señal de entrada es un voltaje que en la pantalla del tubo de rayos catódicos se transforma en luz que nosotros vemos. Todo sería correcto si la luz emitida fuera directamente proporcional al voltaje que le llega, pero en muchos casos no ocurre esto, sino que la relación es aproximadamente cuadrática. La relación no lineal entre luz emitida y tensión de entrada se puede expresar:

$$B = k\ (V\text{^}gamma)$$

Ecuación 5–17 Función gamma

siendo B la luz, V la tensión '^' signo de potencia y gamma un factor de descompensación y que produce esta no linealidad. Algunos valores típicos de gamma según el tubo de rayos catódicos son 1.7, 2.8 y también los hay de valor 1 que no producen esta alteración.

La corrección gamma lo que hace es elevar la señal al exponente 1/gamma para compensar el efecto comentado anteriormente.

Existen otros muchos efectos de este tipo y otros muchos pueden ser inventados, sobre todo si no se busca una utilidad práctica de los mismos sino únicamente una utilidad estética, así que se trata de un interesante campo de experimentación. La forma más fácil de jugar con nuevos efectos es aplicando diferentes funciones de transferencia a las imágenes.

El uso de la función gamma permite en muchos casos reducir las imperfecciones de las imágenes como consecuencia de la imperfección de los dispositivos ópticos. Por ello, puede ser

usada como filtro para suavizar imágenes dependiendo del valor de entrada que se le asigne a la constante.

Las gráficas de la función gamma para un histograma cualquiera podrían ser:

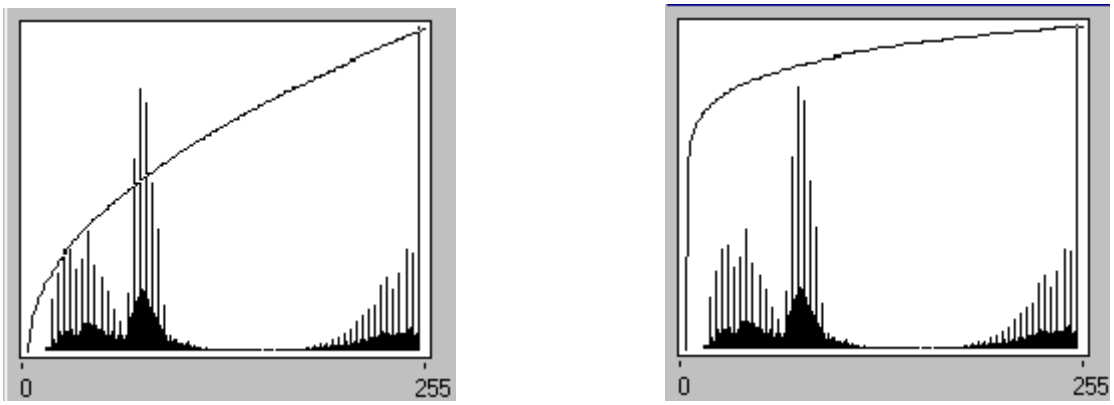


Figura 5–35 Gráfica de la función gamma para gamma = 2 y gamma = 10 respectivamente

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Gamma(const ElementT g,const ElementT min, const ElementT max).

  Calcula la función gamma a la matriz de la imagen a partir de los valores del coeficiente de entrada g, del máximo y mínimo que puede tomar la imagen.

- ***Parámetros de entrada***

  g coeficiente de la función gamma.

  min mínimo posible a tomar por la imagen.

  max máximo posible a tomar por la imagen.

- ***Parámetros de salida***

  matriz de la imagen de salida con la función gamma aplicada.

El procedimiento implementa el filtro de la función gamma sobre una matriz. Se basa en aplicar un valor constante de entrada *g*, a partir de la función descrita con anterioridad, de manera que para cada píxel se le aplique la potencia de *1/g*.

Ejemplo

```
C_Image image;
C_Matrix::ElementT g=1;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);
```

```
C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);

C_Trace("Calculating Gamma Image");
image.Gamma(g,0,(colorsN-1));

C_Trace ("Writting Gamma Image...");
image.WriteBMP (OutFile);

image.Free();
```

Resultados

Los resultados obtenidos son los que se muestran a continuación:

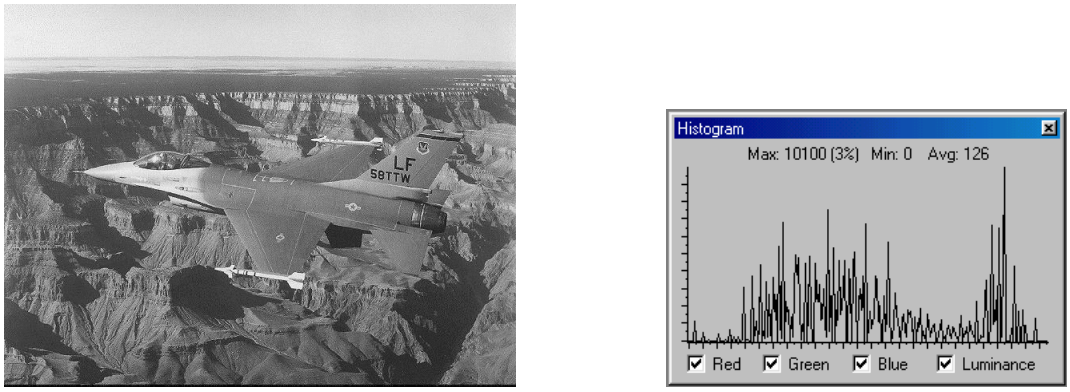


Figura 5–36 Imagen original ‘aviongr.bmp’ y su histograma

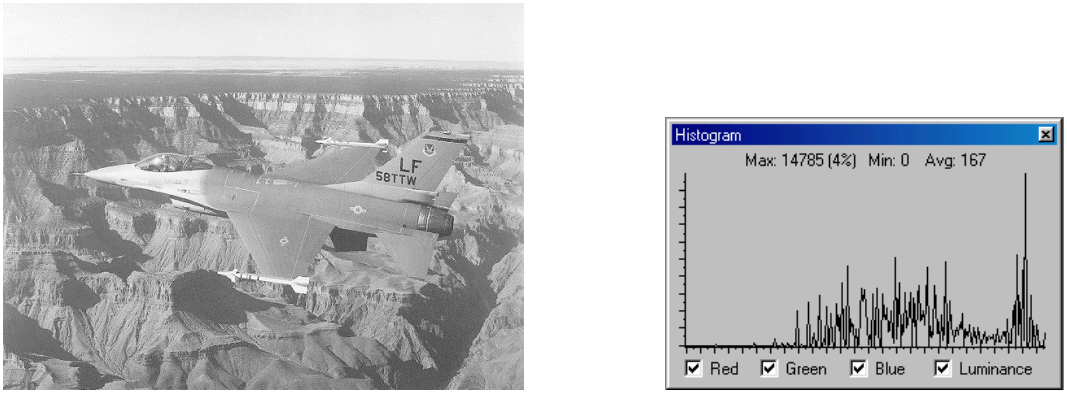


Figura 5–37 Imagen gamma = 2 ‘aviongr.bmp’ y su histograma

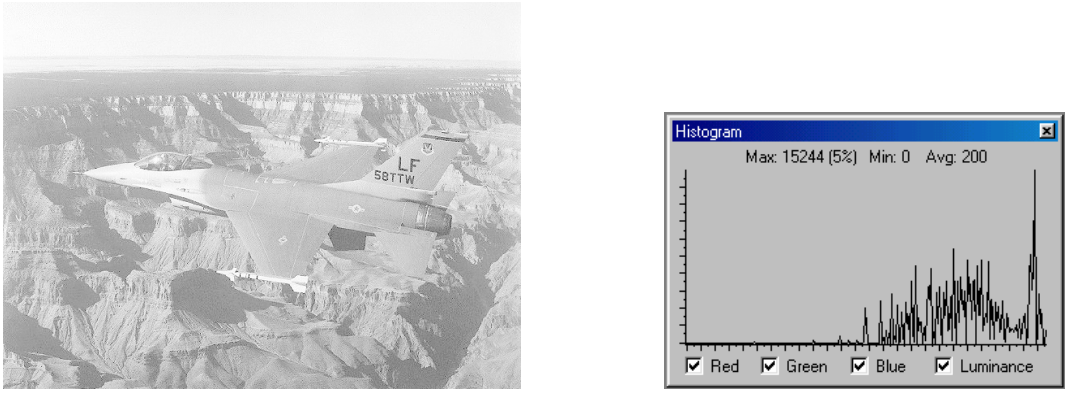


Figura 5–38 Imagen gamma = 10 ‘aviongr.bmp’ y su histograma

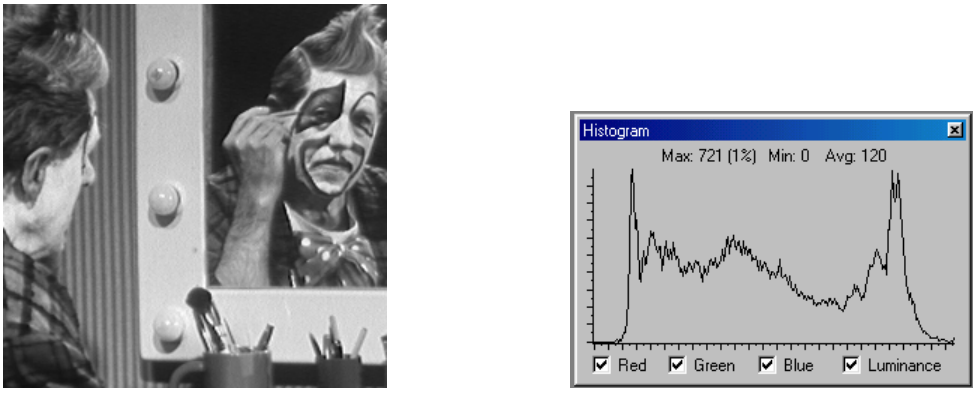


Figura 5–39 Imagen original ‘payaso.bmp’ y su histograma

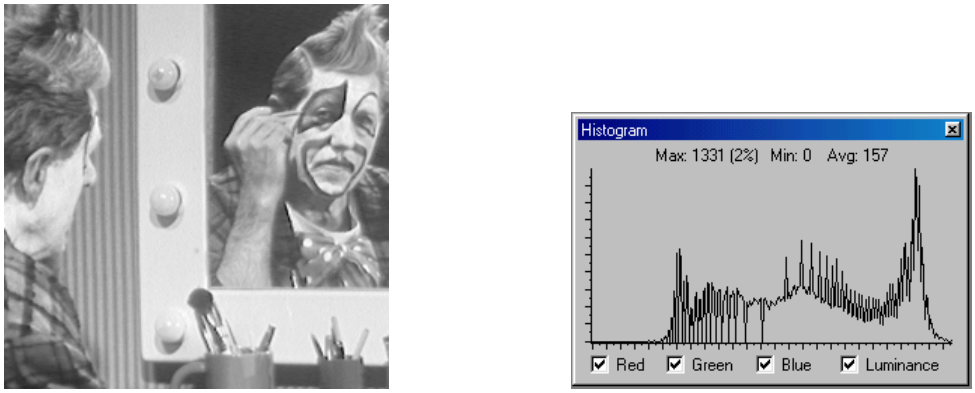


Figura 5–40 Imagen gamma = 2 ‘payaso.bmp’ y su histograma

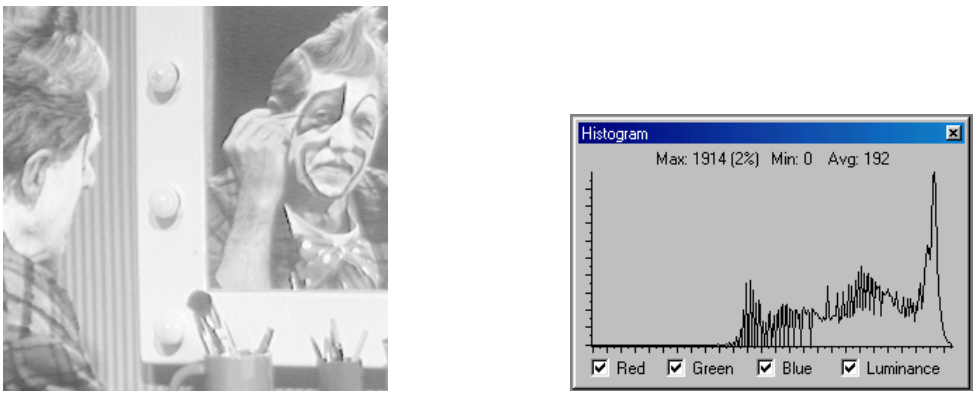


Figura 5–41 Imagen gamma = 10 ‘payaso.bmp’ y su histograma

Los resultados obtenidos a continuación expresan la característica de que a medida que se incrementa el valor de g el histograma de la imagen tiende hacia los niveles más claros de la imagen. Es lógico, ya que si se contemplan las graficas anteriores se ve que en función del valor de g (constante gamma) la función tiende a tomar valores altos correspondientes en una imagen con niveles de gris a los más claros.

5.3.7 Función Uniforme

Sea X una distribución uniforme [5] en un intervalo [a, b], si toma valores en ese intervalo su función de densidad se puede definir como:

$$f(x) = \begin{cases} \dfrac{1}{(b-a)} & Si\ a < x < b \\ 0 & En\ el\ resto \end{cases}$$

La función uniforme aplicada a una imagen permite darle una especie de suavizado.

La gráfica que la representa para un histograma cualquiera es la siguiente:

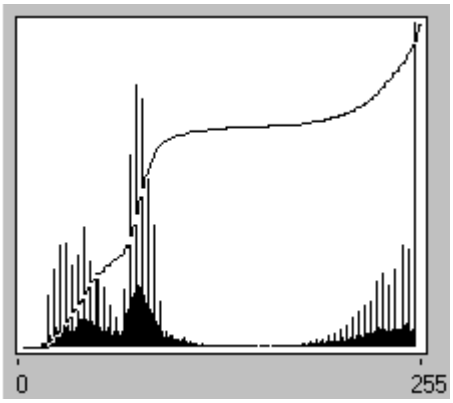


Figura 5–42 Función uniforme aplicada a un histograma

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Uniform(const ElementT min, const ElementT max).

  Calcula la función uniforme a la matriz de la imagen a partir de los valores de entrada del máximo y mínimo que puede tomar la imagen

- ***Parámetros de entrada***

  min mínimo posible a tomar por la imagen.

  max máximo posible a tomar por la imagen.

- ***Parámetros de salida***

  matriz de la imagen de salida con la función gamma aplicada.

Para realizar el cálculo de la función uniforme se calcula el rango de valores en el que se encuentra la imagen. A continuación se le aplica la función de distribución que sigue una uniforme, es decir, 1/(Max-Min) por el valor del píxel de la imagen que se esté tratando en ese instante.

El operador de la función uniforme permite suavizar la imagen de acuerdo con los valores máximos y mínimos alcanzados en ella. De esta manera si la imagen es muy oscura con pequeñas regiones claras estas zonas tenderán a oscurecer, para el caso contrario se producirá el efecto inverso.

Ejemplo

```
C_Image image;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);

C_Trace("Calculating Uniform Image");
image.Uniform(0,(colorsN-1));

C_Trace ("Writting Uniform Image...");
image.WriteBMP (OutFile);

image.Free();
```

Resultados

Los resultados obtenidos para las distintas imágenes son los siguientes:

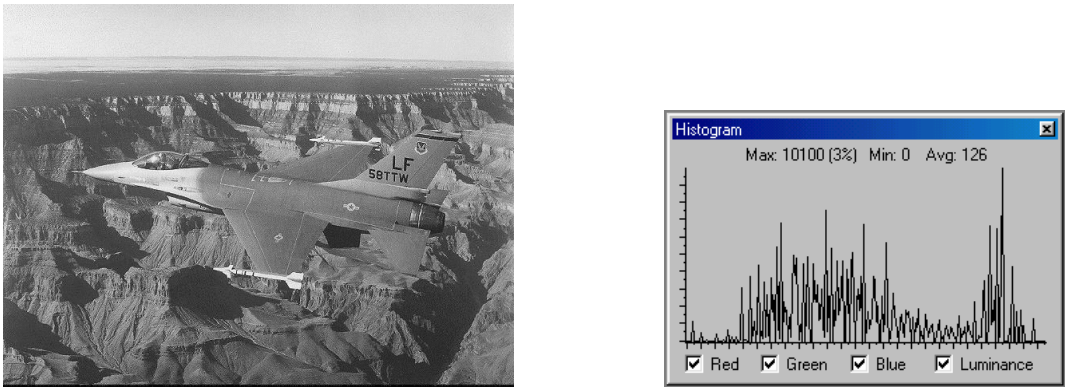


Figura 5–43 Imagen original ‘aviongr.bmp’ y su histograma

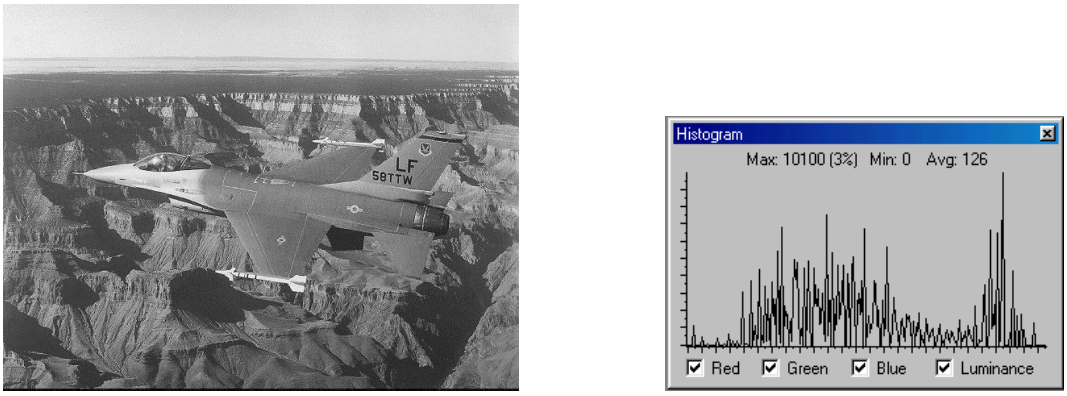


Figura 5–44 Imagen uniforme ‘aviongr.bmp’ y su histograma

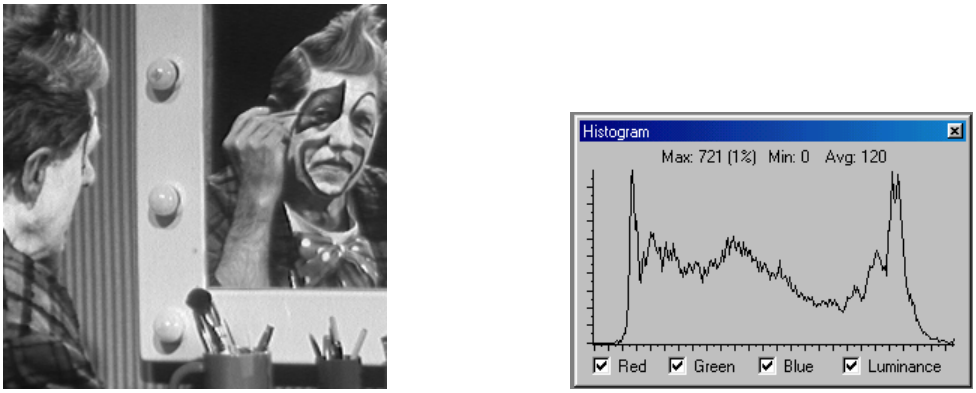


Figura 5–45 Imagen original ‘payaso.bmp’ y su histograma

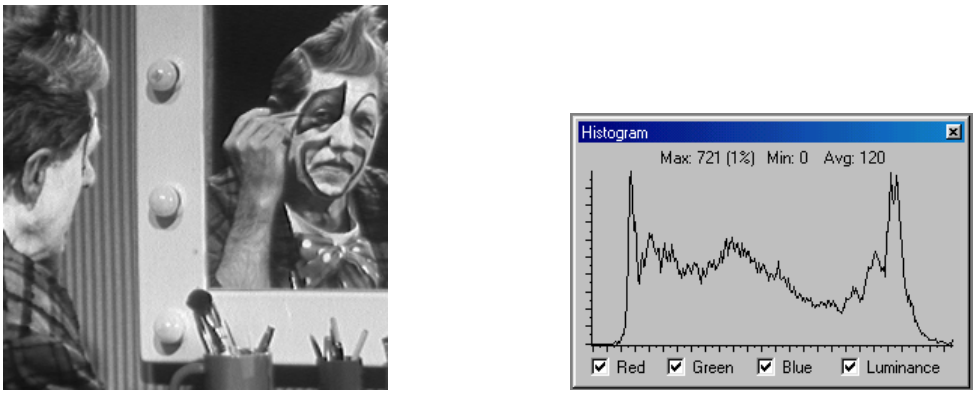


Figura 5–46 Imagen uniforme ‘payaso.bmp’ y su histograma

Para cada uno de los dos ejemplos se produce una leve modificación, simplemente en algunos picos muy altos del histograma se modifica un poco. Esto también es debido a que la constante de multiplicación es muy pequeña lo que produce una diferenciación menor.

5.4 MÉTODOS DE PROCESAMIENTO DE MÁSCARAS

5.4.1 Aplicación de matrices de convolución

Para la aplicación de la gran mayoría de filtros o transformaciones a la imagen se suele utilizar frecuentemente una matriz de convolución o máscara [2]. Esta ha sido comentada brevemente en la introducción del capítulo, a continuación se definirá de manera precisa.

Una matriz de convolución es una tabla que cumple la restricción de que su tamaño debe ser $2^n + 1$, donde n = 1, 2, 3,... A continuación se explica la manera de realizar los cálculos para una matriz de convolución de tamaño 3×3.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Tabla 5–1 Matriz de convolución

z_1	z_2	z_3	z_4	z_5	z_6
z_7	z_8	z_9	z_{10}	z_{11}	z_{12}
z_{13}	z_{14}	z_{15}	z_{16}	z_{17}	z_{18}
z_{19}	z_{20}	z_{21}	z_{22}	z_{23}	z_{24}
z_{25}	z_{26}	z_{27}	z_{28}	z_{29}	z_{30}
z_{31}	z_{32}	z_{33}	z_{34}	z_{35}	z_{36}

Tabla 5–2 Matriz de una imagen en niveles de gris

Cada uno de los elementos de la tabla 1 representa el coeficiente o peso asignado en la matriz de convolución. En cuanto a la tabla 2 representa cada uno de los elementos a los niveles de gris de la imagen. El cálculo de la matriz de convolución aplicado a un píxel por ejemplo z_{15} sería el siguiente.

$$z'_{15} = z_8 w_1 + z_9 w_2 + z_{10} w_3 + z_{14} w_4 + z_5 w_5 + z_{16} w_6 + z_{20} w_7 + z_{21} w_8 + z_{22} w_9$$

En otras palabras, se superpone la matriz de convolución sobre el píxel que se quiere analizar tomando a este como centro. Para cada uno de los elementos superpuesto se efectúa el producto de ambos y posteriormente se realiza la suma de todos los productos obtenidos. Este cálculo plantea el problema de aplicar la matriz de convolución a un píxel que se encuentre en un borde de la imagen.

Para resolver el problema planteado hay que determinar primero cuales son las regiones o zonas conflictivas a resolver. Mostrar como se resolvería para el caso de una matriz de convolución de dimensiones 3 × 3 y posteriormente generalizarlo para matrices de convolución de tamaño N × N, teniendo en cuenta la restricción comentada al principio del apartado.

Las regiones problemáticas de una imagen a la hora de aplicar una matriz de convolución son las que se muestran en la siguiente figura.

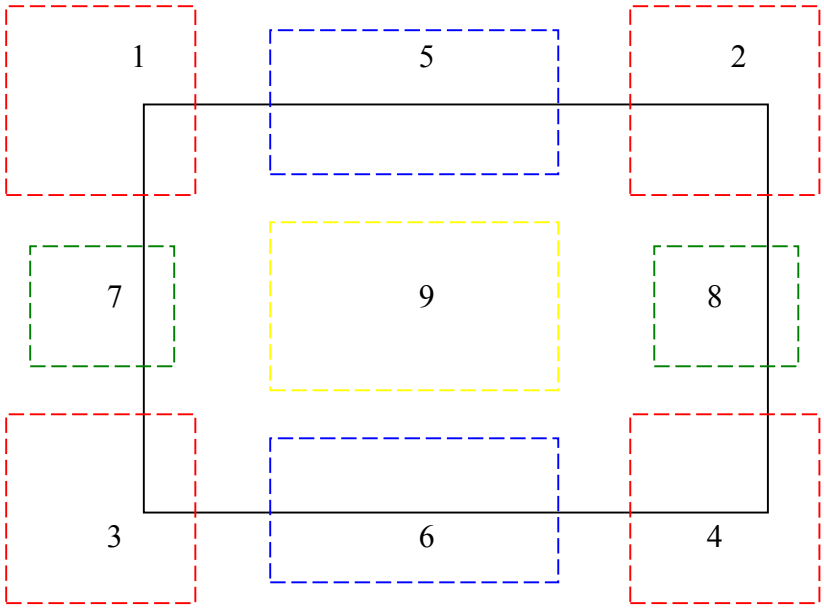


Figura 5–47 Regiones en el tratamiento de bordes

Cada una de las regiones representa un tipo de borde a calcular:

- Región1: esquina superior izquierda.
- Región 2: esquina superior derecha.
- Región 3: esquina inferior izquierda.
- Región 4: esquina inferior derecha.
- Región 5: lateral superior.
- Región 6: lateral inferior.
- Región 7: lateral izquierdo.
- Región 8: lateral derecho.
- Región 9: centro.

Estas regiones, salvo la número 9 cuyo cálculo ha sido descrito, presentan una serie de problemas y criterios a tener en cuenta. A continuación, se va a describir como se resuelve para cada región en función de una matriz de convolución de tamaño 3 × 3 y como se realizaría para tamaños más grandes.

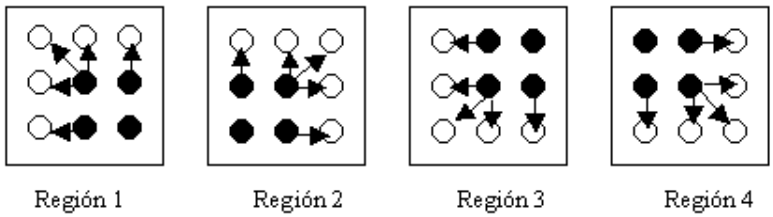


Figura 5–48 Tratamiento de las esquinas

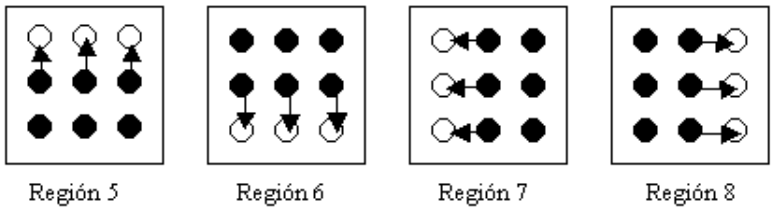


Figura 5–49 Tratamiento de los laterales

En este caso los píxeles en negro corresponden a valores de la imagen y los píxeles en blanco a puntos fuera de la imagen y que no contienen valor. El procedimiento se basa en asignar un valor a los píxeles en blanco siguiendo algún criterio. La forma de asignación se efectúa de manera que un píxel del borde de la imagen siempre va a asignar un valor a su adyacente en blanco. Para una matriz de convolución de mayor tamaño se daría el caso de que la asignación no sería a un elemento adyacente en blanco sino que podría haber más de uno al que asignar el valor. En este caso, se copiaría el valor de los elementos de la misma dirección en la que establecen las flechas con la salvedad que habrá más elementos a copiar.

Otra posible solución sería copiar los valores de regiones opuestas, es decir, contemplar que las imágenes son circulares. Esto se deduce del dominio de frecuencias, donde una vez que la función de la imagen se encuentra en el dominio de las frecuencias, esta es considerada periódica. Este concepto de periodicidad es el que da sentido a la solución propuesta. El problema es que en la realidad las imágenes no son periódicas y no tienen por que coincidir regiones opuestas. Esto expresa que no sería la solución más idónea.

En cuanto a la implementación es necesario establecer un conjunto de procedimientos para analizar, no solo en la región que se encuentra sino también cada uno de los procedimientos necesarios para trabajar con matrices de tamaño N × N.

El método global que implementa la convolución es el siguiente:

La interfaz de acceso al procedimiento

- ***Procedimiento***

  CalculateConvolution(C_Matrix matrix, C_Matrix convMatrix).

  Calcula la matriz de la imagen mediante la aplicación de una matriz de convolución de entrada.

- ***Parámetros de entrada***

  matrix matriz de la imagen de entrada.

  conMatrix matriz de convolución.

- ***Parámetros de salida***

  matriz de la imagen de salida con la aplicación de la convolución.

Ejemplo

```
int c=1;
C_Image image,image2;
C_Matrix conv;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Trace("Reading Convolution...");
conv.Read(ConvFile);

image2 = image;
C_Trace("Filtering Convolution...");
image.CalculateConvolution(image2,conv);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
image.Stretch(0,(colorsN-1));

C_Trace ("Writting Filter Image...");
image.WriteBMP (OutFile);

conv.Free();
image.Free();
image2.Free();
```

Para los módulos que extraen submatrices de las esquinas el comportamiento de manera genérica puede enfocarse de la siguiente manera:

- **Copiar la imagen.** Se basa en copiar las posiciones de los píxeles que coincidan sobre la matriz de la imagen en la matriz de convolución.

- **Copiar la esquina.** Se corresponde con coger el píxel con contenido de la esquina en el que a continuación los adyacentes que se encuentren sean los que no poseen valores. Se copia el contenido del mismo a partir de las direcciones establecidas en la Figura 5–48.

- **Copiar ambos laterales.** Se centra en copiar los píxeles con contenido en los adyacentes sin contenido mediante los dibujos establecidos en la Figura 5–49.

Por otra parte, los módulos que extraen submatrices de los laterales el comportamiento general que puede extraerse de ellos es el siguiente:

- **Copiar la imagen.** Se basa en copiar las posiciones de los píxeles que coincidan sobre la matriz de la imagen en la matriz de convolución.

- **Copiar lateral.** Se centra en copiar los píxeles con contenido en los adyacentes sin contenido mediante los dibujos establecidos en la Figura 5–49.

El conjunto de módulos que extraen submatrices es independiente del tamaño de la matriz de convolución, de ahí su complicación a la hora de generar un módulo que funcione para cualquier valor del tamaño (siempre que cumpla las restricciones del tamaño establecidas al principio del apartado). Si se encuentra el recorrido de la convolución por algún punto del centro, no será necesario extender los valores de la matriz, ya que simplemente se procederá a copiar los valores de la matriz de la imagen a la submatriz.

Llegando a este punto en el tratamiento del centro de la matriz vamos a disponer del resultado obtenido al aplicar la matriz de convolución según el tamaño. De este modo podemos aplicar cualquier filtro de convolución simplemente leyendo la matriz de datos del archivo que la almacena, asignar dichos datos a una matriz de convolución y aplicar la convolución a la matriz de la imagen para obtener el resultado de la aplicación de diversos filtros.

No obstante, existe la posibilidad de necesitar que se aplique más de una matriz de convolución como es el caso de los operadores de Sobel o Prewitt, donde se necesitan dos matrices. Para facilitar los cálculos se ha desarrollado el método que permite aplicar la convolución sobre dos matrices de manera simultánea.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  CalculateConvolution2(C_Matrix matrix, C_Matrix Gx, C_Matrix Gy).

  Aplica dos matrices de convolución a la matriz de entrada y la almacena en la matriz que referencia al módulo.

- ***Parámetros de entrada***

  matrix matriz de entrada.

  Gx matriz de convolución.

  Gy matriz de convolución.

- ***Parámetros de salida***

  matriz de salida con la convolución calculada para las dos matrices.

Ejemplo

```
int c=1;
C_Image image,image2;
C_Matrix Gx,Gy;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Trace("Reading Convolution Gx...");
Gx.Read(FileGx);

C_Trace("Reading Convolution Gy...");
Gy.Read(FileGy);

image2 = image;
C_Trace("Filtering Two Matrix of Convolution ...");
image.CalculateConvolution2(image2,Gx,Gy);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
image.Stretch(0,(colorsN-1));

C_Trace ("Writting Filter Two Matrix of Convolution Image...");
image.WriteBMP (OutFile);

Gx.Free();
Gy.Free();
image.Free();
image2.Free();
```

Además, puede darse la necesidad de aplicar cuatro de matrices de convolución a la vez. El caso más común es la detección de bordes mediante cuatro matrices de convolución que

habilitan las diferentes direcciones. Por lo que se ha creído conveniente desarrollar otro módulo que se muestra a continuación.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  CalculateConvolution4(C_Matrix matrix, C_Matrix Gx,C_Matrix Gy, C_Matrix Gdp,C_Matrix Gds).

  Aplica cuatro matrices de convolución a la matriz de entrada y la almacena en la matriz que referencia al módulo.

- ***Parámetros de entrada***

  matrix matriz de entrada.

  Gx matriz de convolución.

  Gy matriz de convolución.

  Gdp matriz de convolución.

  Gds matriz de convolución.

- ***Parámetros de salida***

  matriz de salida con la convolución calculada para las cuatro matrices.

Ejemplo

```
int c=1;
C_Image image,image2;
C_Matrix convH,convV,convdp,convds;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Trace("Reading Convolution Horizontal...");
convH.Read(FileH);
C_Trace("Reading Convolution Vertical...");
convV.Read(FileV);
C_Trace("Reading Convolution -45°...");
convdp.Read(Filedp);
C_Trace("Reading Convolution +45°...");
convds.Read(Fileds);

image2 = image;
C_Trace("Filtering Four Matrix of Convolution ...");
image.CalculateConvolution4(image2,convH,convV,convdp,convds);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
```

```
image.Stretch(0,(colorsN-1));

C_Trace ("Writting Filter Four Matrix of Convolution Image...");
image.WriteBMP (OutFile);

convH.Free();
convV.Free();
convdp.Free();
convds.Free();
image.Free();
image2.Free();
```

Resultados

Los distintos resultados de este apartado se muestran de manera más representativa en cada uno de los siguientes apartados reflejando diferentes aplicaciones de matrices de convolución sobre las imágenes. Se muestra una visualización y conceptos extraídos de la aplicación para los distintos filtros descritos a continuación.

5.4.2 Filtro de la media

El **filtro de la media** es un [5] simple, intuitivo y fácil método de implementación de igualar los píxeles de una imagen, por ejemplo, reduciendo la diferencia de intensidad entre un píxel y sus vecinos. Este método se utiliza muy a menudo para reducir el ruido de las imágenes.

La idea del filtro de la media es reemplazar el valor de cada píxel en una imagen por la media aritmética de los valores de sus vecinos, incluyendo a él mismo. Esto produce el efecto de eliminación de los valores de aquellos píxeles no representativos del contorno. El filtro de la media es aplicado como un filtro de convolución. Como todos las convoluciones se aplica en torno a un valor denominado kernel, que representa la forma y tamaño de los vecinos que van a ser utilizados cuando se calcula la media.

Normalmente se utiliza una matriz de 3×3 para aplicar este filtro. Sin embargo, matrices de 5×5 pueden usarse para una igualación de los píxeles de una imagen más severa.

La figura siguiente representa una matriz de 3×3 utilizada para el cálculo de la media de los píxeles de una imagen:

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Tabla 5–3 Matriz de convolución para el cálculo de la media

Para la creación de un método que implemente el filtro de la media necesitamos recibir como parámetro el tamaño de la matriz del entorno. Además, necesitamos de unas subrutinas que apliquen la matriz del entorno a determinadas regiones de la imagen del mismo tamaño que la matriz de entorno.

Este método recibe como parámetro el tamaño de la matriz del entorno y calculando la media para cada píxel de la imagen de sus vecinos incluyendo también al mismo píxel. Para ello debemos definir una pequeña matriz de tamaño **sizeConv x sizeConv** sobre la que se calculará la media. El valor medio resultante de esta pequeña matriz será el valor que se le asigne al píxel de la imagen que se este recorriendo en cada paso.

La función **AssingSubMatrix()** obtiene la pequeña submatriz sobre la que se calculará la media. Esta función debe ser capaz de diferenciar la posición del píxel tratado en la imagen. La tarea realizada por esta función se explica detalladamente en el apartado correspondiente a la aplicación de matrices de convolución.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  MeanFilter(C_Matrix & matrix, IndexT sizeConv).

  Calcula la media para cada uno de los píxeles de la matriz de entrada en función del tamaño de la matriz del entorno.

- ***Parámetros de entrada***

  matrix matriz de entrada.

  sizeConv tamaño de la matriz del entorno a aplicar.

- ***Parámetros de salida***

  matriz de salida.

Ejemplo

```
C_Image image,image2;
C_Matrix::IndexT sizeConv;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

image2 = image;
C_Trace("Mean Filtering ...");
image.MeanFilter(image2,sizeConv);

C_Trace ("Writting Filter Image...");
image.WriteBMP (OutFile);

image.Free();
image2.Free();
```

Resultados

Para mostrar el resultado de aplicar el filtro de la media presentaremos los resultados obtenidos para ambas imágenes originales y para las matrices de 3x3 y 7x7, así como su histograma correspondiente. Los ejemplos mostrados ilustran dos de los problemas principales del filtro de la media:

- Un simple píxel con un valor muy poco representativo afecta al valor de la media de todos los píxeles que son sus vecinos.

- Cuando se encuentra un borde expandido en los vecinos del píxel tratado, el filtro interpolará nuevos valores para los píxeles del borde y así se difuminará el borde. Esto puede ser un problema si la forma de los bordes es de importancia en la salida de la imagen.

Estos dos problemas son abordados por el filtro de la mediana que veremos en el siguiente apartado. Este es a menudo un mejor filtro para la reducción de ruido que el filtro de la media, pero su cálculo es mas complejo computacionalmente. Dicho esto, mostramos los resultados obtenidos a continuación.

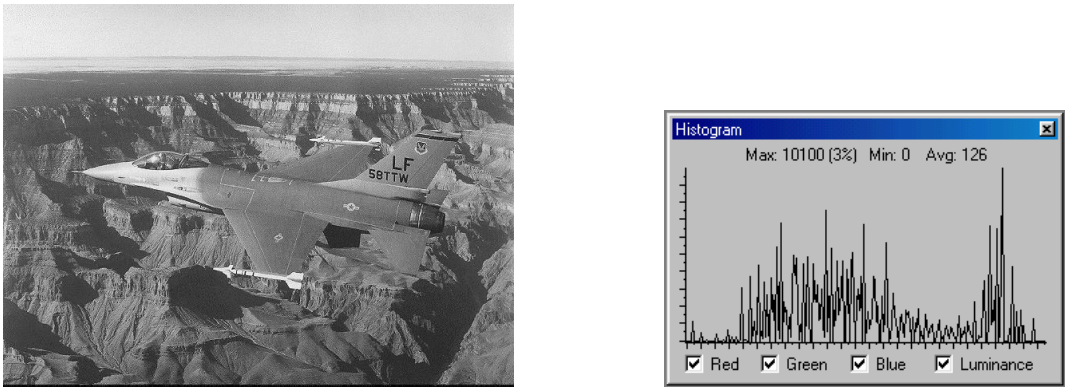


Figura 5–50 Imagen original ‘aviongr.bmp’ y su histograma

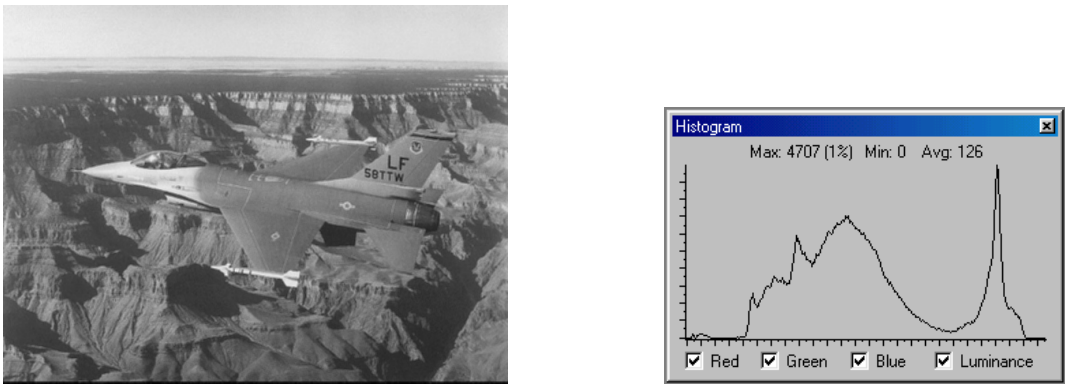


Figura 5–51 Imagen filtrada por la media (3 × 3) ‘aviongr.bmp’ y su histograma

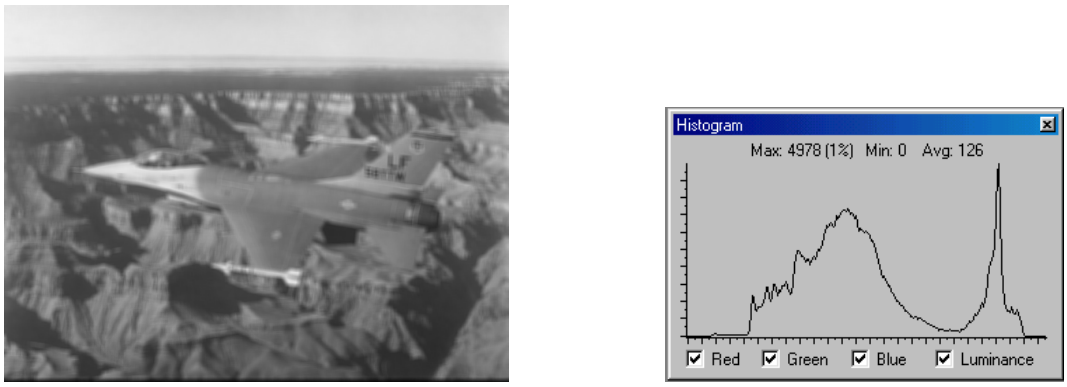


Figura 5–52 Imagen filtrada por la media (7 × 7) ‘aviongr.bmp’ y su histograma

Se puede apreciar como se produce un difuminado de la imagen eliminando ruido de la imagen original. También podemos ver como una matriz de 7×7 produce una eliminación mayor del ruido de la imagen. Además, simplemente con ir visualizando el histograma se observa como da una mayor suavidad al contorno de la misma a medida que crece la matriz de convolución.

La siguiente figura muestra el resultado obtenido para la otra imagen original:

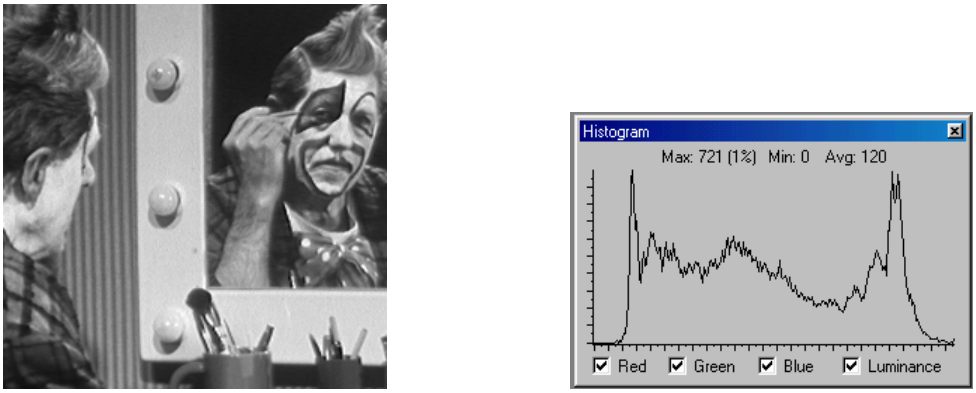


Figura 5–53 Imagen original ‘payaso.bmp’ y su histograma

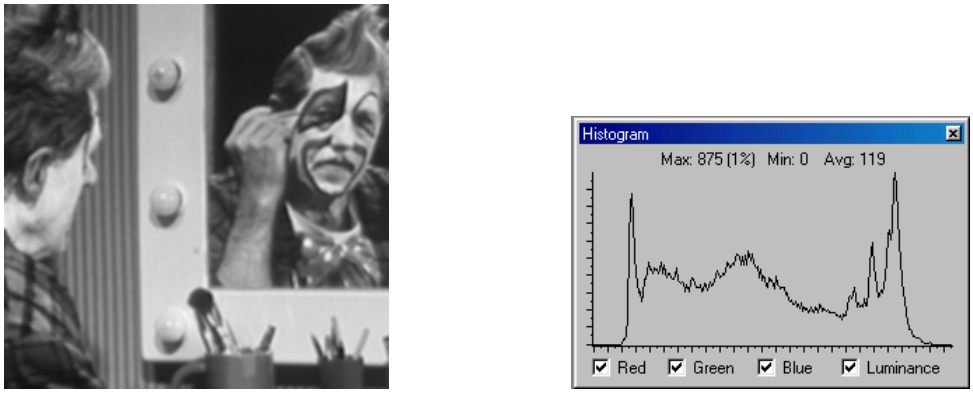


Figura 5–54 Imagen filtrada por la media (3 × 3) ‘payaso.bmp’ y su histograma

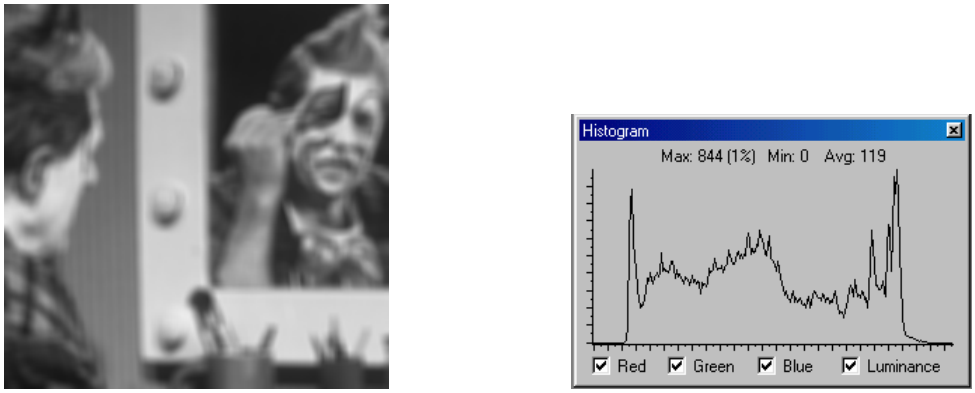


Figura 5–55 Imagen filtrada por la media (7 × 7) ‘payaso.bmp’ y su histograma

En esta segunda imagen donde los contornos son mas diferenciados, se aprecía notablemente el problema comentado anteriormente. La aplicación del filtro de la media para la eliminación de ruido expande los contornos de la imagen. En los histogramas es donde mejor se aprecia la variación que sufre. A medida que la matriz de convolución crece realiza como un estiramiento del mismo intentando eliminar los sucesivos picos de niveles de gris producidos en la imagen. Esto es muy apreciable en la Figura 5–55. donde el histograma inicial es muy inconsistente, en cuanto al gran número de protuberancias que contiene y en la medida que se incrementa el tamaño de la matriz de convolución estas protuberancias van disminuyendo. Se

produce no solo un suavizado en los niveles de gris de la imagen sino en la propia función del histograma.

5.4.3 Filtro de la mediana

En el apartado anterior se había descrito el problema de que la media difumina los bordes y otros detalles de realce de la imagen. Cuando el objetivo es más la reducción del ruido que el difuminado, el empleo de los filtros de mediana [2, 5]es uno de los métodos a adoptar. El procedimiento se centra en reemplazar el nivel de gris de cada píxel por la mediana de los niveles de gris en un entorno de este píxel definido en función de un tamaño. Este método actúa bien cuando el ruido está formado por elementos en cantidad y de cuya figura del histograma es puntiaguda, así como se desea conservar el contorno de las distintas regiones.

La mediana de un conjunto de valores es tomar el valor central después de aplicar una ordenación de los mismos. Para aplicar la mediana a un píxel, primero se deben extraer los valores del píxel y sus vecinos en una matriz de tamaño definida donde el píxel a tratar se encuentra en el centro de la misma. Una vez extraído el conjunto de valores se ordenan de mayor a menor o viceversa, según el criterio establecido, y a continuación se toma el valor central del conjunto de ordenación. Así la función principal del filtrado por la mediana consiste en introducir puntos con diferentes intensidades que sean más parecidos a sus vecinos, eliminando de esta forma los estrechos picos de intensidad que aparecen aislados en la máscara tomada.

Para entender el funcionamiento basta con poner un simple ejemplo. Imaginemos que un entorno 3×3 presenta los siguientes valores:

1	2	2
2	**5**	2
2	7	10

A continuación se ordenan:

1	2	2	2	**2**	2	5	7	10

se toma el elemento central al que corresponde una mediana de 2. De esta forma se observa como los valores más semejantes tienen mayor probabilidad frente a valores aislados.

Al igual que para el filtro de la media a cada píxel de la imagen se le aplica el valor de la mediana resultante de sus píxeles vecinos y el propio píxel, por lo que en el procedimiento que implementa este filtro debemos volver a utilizar las imágenes que obtienen una pequeña submatriz de la matriz de la imagen original.

Los valores resultantes en la matriz se ordenan de menor a mayor mediante la función *Sort()* y se elige el valor que ocupa la posición de la mitad en la matriz.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  MedianFilter(C_Matrix & matrix, IndexT sizeConv).

  Calcula la mediana para cada uno de los píxeles de la matriz de entrada en función del tamaño de la matriz del entorno.

- ***Parámetros de entrada***

  matrix matriz de entrada.

  sizeConv tamaño de la matriz del entorno a aplicar.

- ***Parámetros de salida***

  matriz de salida.

El cálculo computacional para el filtro de la mediana es mas complejo siendo el tiempo de ejecución mayor para el cálculo de la mediana que para el cálculo de la media ya que se debe ordenar la matriz del entorno antes de asignar el valor al píxel tratado.

Ejemplo

```
C_Image image,image2;
C_Matrix::IndexT sizeConv;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

image2 = image;
C_Trace("Median Filtering ...");
image.MedianFilter(image2,sizeConv);

C_Trace ("Writting Median Filter Image...");
image.WriteBMP (OutFile);

image.Free();
image2.Free();
```

Resultados

Presentaremos a continuación los resultados obtenidos al aplicar el filtro de la mediana a las imágenes originales con matrices de 3x3 y 5x5 para comparar ambos resultados para las dos imágenes.

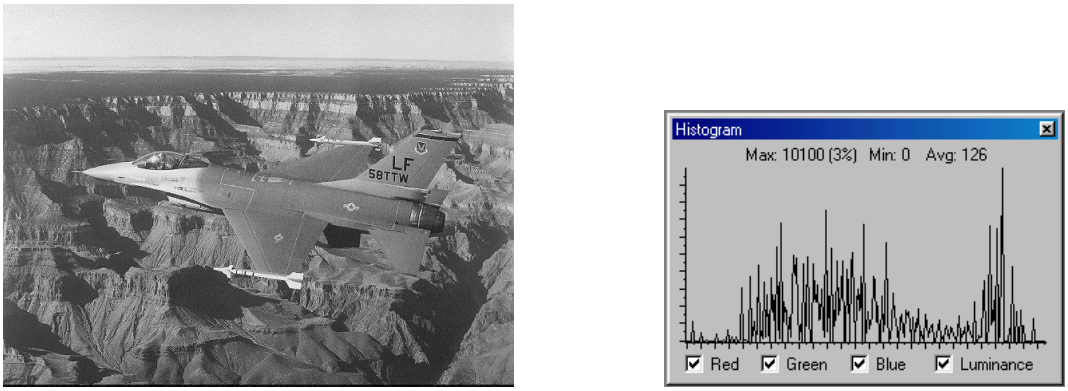


Figura 5–56 Imagen original ‘aviongr.bmp’ y su histograma

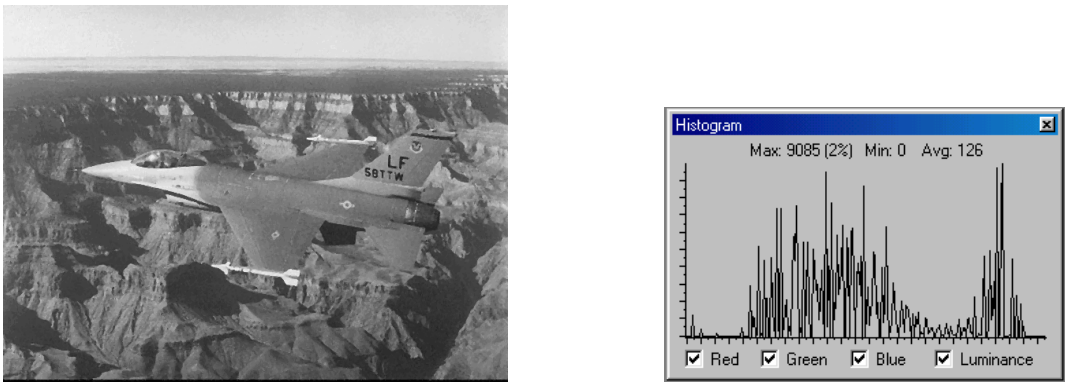


Figura 5–57 Imagen filtrada por la mediana (3 × 3) ‘aviongr.bmp’ y su histograma

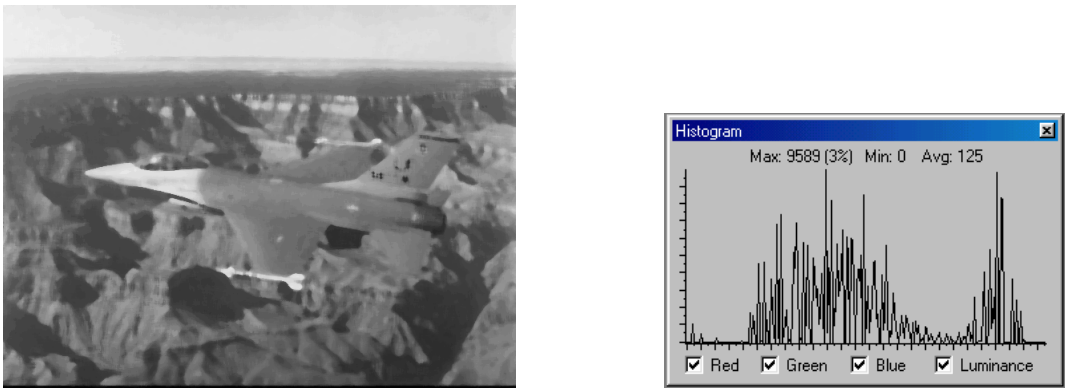


Figura 5–58 Imagen filtrada por la mediana (7 × 7) ‘aviongr.bmp’ y su histograma

Las imágenes resultantes son el producto de la eliminación de componentes de nivel de gris aislados en función de sus vecinos. Donde mejor se observa es en el histograma para niveles de gris con poca aparición, cuando se les aplica el método estos son prácticamente borrados. Por otra parte, a medida que se incrementa el tamaño de la matriz del entorno a aplicar al píxel, se observa como la función del histograma decrece y se ensancha. Esto se debe a que los píxeles están muy influenciados por sus adyacentes.

La siguiente figura muestra el resultado obtenido para la otra imagen original:

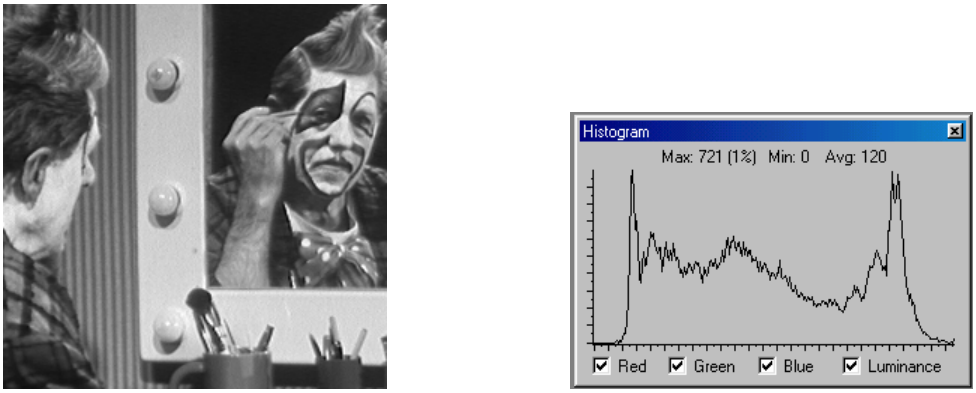


Figura 5–59 Imagen original ‘payaso.bmp’ y su histograma

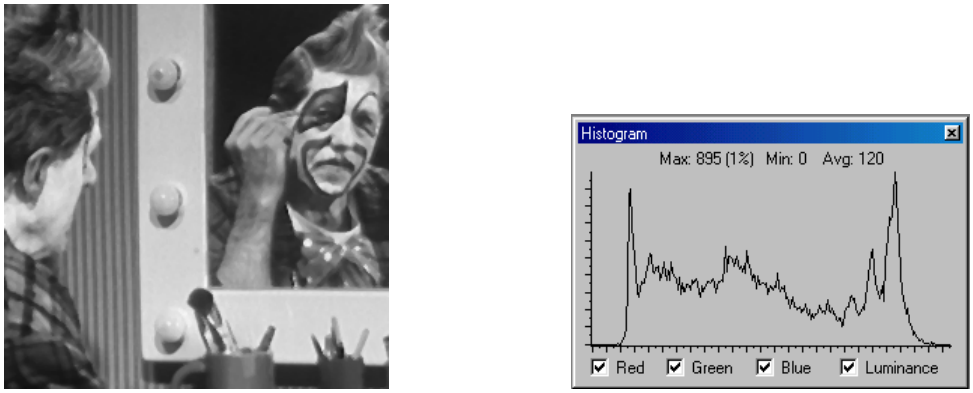


Figura 5–60 Imagen filtrada por la mediana (3 × 3) ‘payaso.bmp’ y su histograma

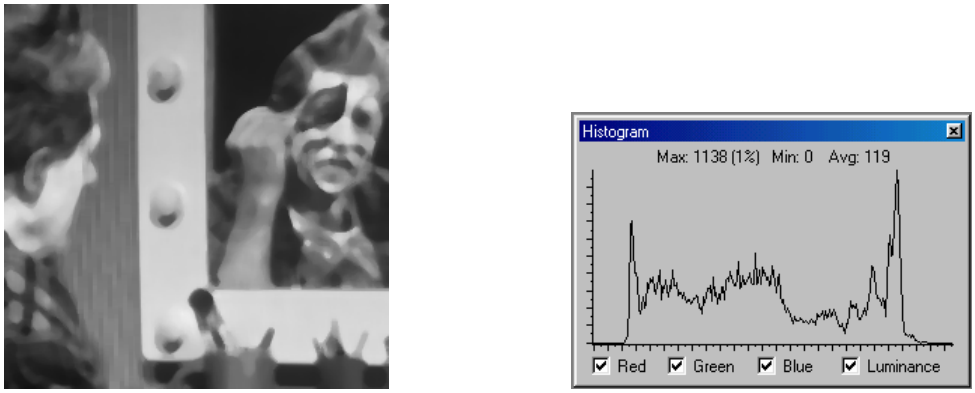


Figura 5–61 Imagen filtrada por la mediana (7 × 7) ‘payaso.bmp’ y su histograma

Al igual que para la figura del avión los resultados obtenidos son idénticos. En ambos casos se aprecia una reducción del histograma, es decir, tiende a compactarse. Los comentarios descritos anteriormente valen para esta figura.

La superioridad del filtro de mediana sobre la media es bastante evidente comparando los resultados. No obstante, dependiendo siempre del dominio del problema. Con el filtro de la mediana eliminas el ruido pero no las sombras, en cambio la media te permite difuminar

eliminando la sobra de los objetos sobre superficies claras. Obviamente si la superficie del fondo es negra no existirán sombras.

El cálculo del valor de la mediana presenta las siguientes ventajas frente al filtro de la media.

- La mediana es un promedio mas robusto que la media, así que un simple píxel no representativo en un vecino no afectara para el valor de la mediana de manera significativa.

- El valor de la mediana debe ser un valor de alguno de los píxeles vecinos. El filtro de la mediana no crea nuevos valores irreales cuando el filtro expande un contorno. Por esta razón el filtro de la mediana es mucho mejor para preservar los bordes y contornos que el filtro de la media.

5.4.4 Filtro paso bajo

El filtro de paso bajo [2, 3, 4]queda englobado dentro de los filtros conocidos como filtros suavizantes. Estos filtros permiten, al igual que el filtrado de la mediana y de la media, hacer que la imagen aparezca algo borrosa, así como eliminar el ruido. Normalmente, suele facilitar la labor antes de la fase de segmentación para la extracción de un objeto.

La restricción necesaria a la hora de un filtro espacial de paso bajo es que la matriz de convolución ha de tener todos los coeficientes positivos. Aunque la forma del filtro espacial pueda ser descrita, por ejemplo, por una función *gaussiana* predeterminada, el requisito clave es que todos los coeficientes sean positivos. La construcción más simple de un filtro espacial paso bajo consistiría en una máscara en la que todos los coeficientes tomasen la unidad como valor. El problema se plantea a la hora del resultado, ya que puede darse el caso en el que la suma de los niveles de gris de los píxeles se saliese del rango válido de grises. Una de las posibles soluciones sería cambiar la escala de la suma dividendo el resultado por el tamaño de la máscara N. Otra solución es tomar en la matriz de convolución cada uno de los coeficientes como 1 / N.

El empleo de este tipo de máscaras según establecen suele denominarse promediado en el entorno.

1	1	1
1	1	1
1	1	1

a) Matriz
3 × 3

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

b) Matriz *5 × 5*

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

c) Matriz *7 x7*

Tabla 5–4 Filtros espaciales paso bajo

A continuación, se muestra el procedimiento para crear una matriz de convolución del tipo de las expuestas anteriormente. El procedimiento que lleva a cabo la aplicación de las matrices de convolución ha sido descrito en el apartado correspondiente.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Lowpass(const ElementT number).

  Calcula la matriz de convolución a partir de un número de entrada de manera que todos los coeficientes de la matriz sean iguales. Se debe establecer el tamaño de la matriz antes de referenciar al procedimiento.

- ***Parámetros de entrada***

  number número de entrada.

- ***Parámetros de salida***

  matriz de salida.

Ejemplo

```
C_Matrix::IndexT sizeConv=3;
C_Matrix::ElementT number=1;

C_Trace ("Generatting The Convolution Matrix of Low Pass...");
C_Matrix conv(1,sizeConv,1,sizeConv);
conv.Lowpass(number);

C_Trace ("Writting The Convolution Matrix of Low Pass...");
conv.Write(OutFile);

conv.Free();
```

Este ejemplo se limita a calcular y almacenar una matriz de convolución paso bajo a partir de un parámetro de entrada. La aplicación de la matriz de convolución sobre una imagen se puede encontrar en el ejemplo del apartado correspondiente a la aplicación de matrices de convolución.

Resultados

La matriz de convolución aplicada para mostrar los siguientes resultados es:

1	1	1
1	1	1
1	1	1

3×3

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

7×7

Tabla 5–5 Matriz de convolución paso bajo

Se ha tomado esta matriz de convolución de dimensiones 7×7 para comprobar la eficiencia de la misma cuando la máscara barre la imagen.

En los resultados que se muestran a continuación se observan como realiza un suavizado sobre la imagen dejando una superficie borrosa o difuminada. Para el histograma de cada figura es donde se aprecia el contraste con mayor medida. Es decir, se produce un suavizado de la curva que define el histograma.

Los resultados obtenidos para las distintas imágenes son los siguientes:

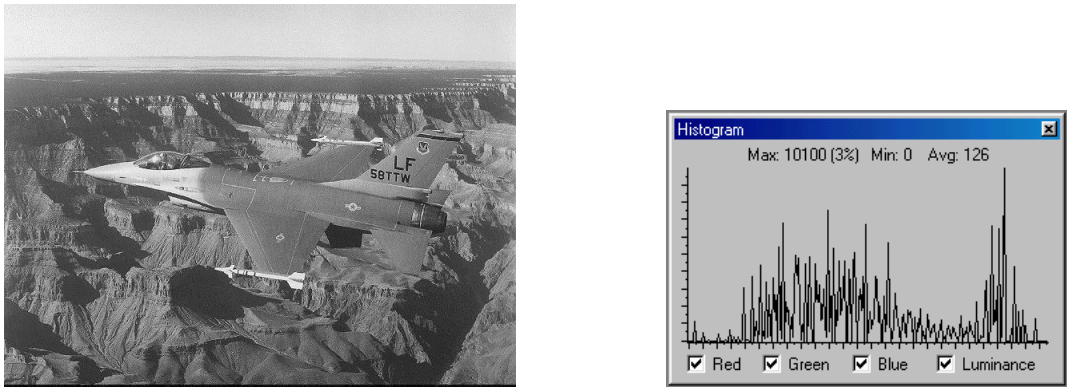


Figura 5–62 Imagen original ‘aviongr.bmp’ y su histograma

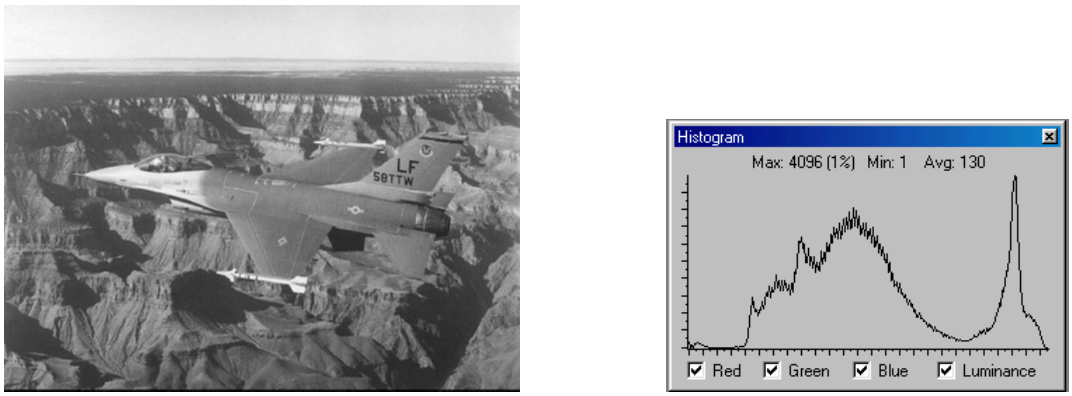


Figura 5–63 Imagen filtrada paso bajo (3 × 3) ‘aviongr.bmp’ y su histograma

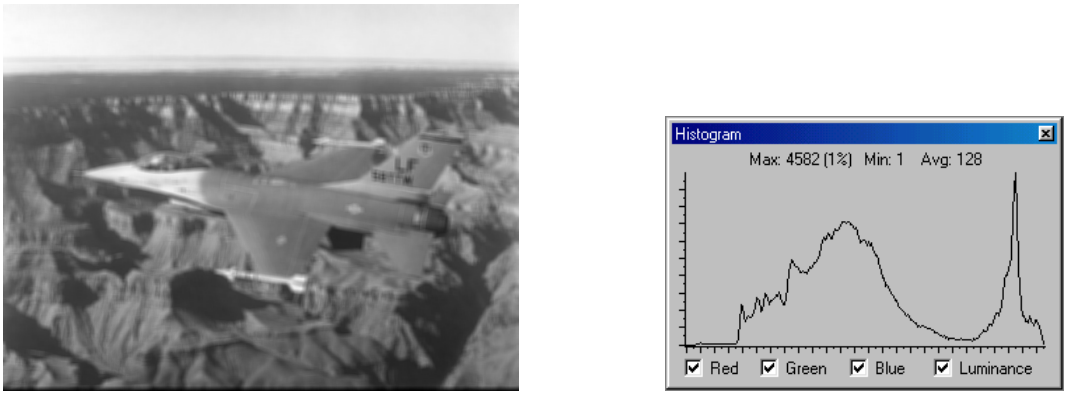


Figura 5–64 Imagen filtrada paso bajo (7 × 7) ‘aviongr.bmp’ y su histograma

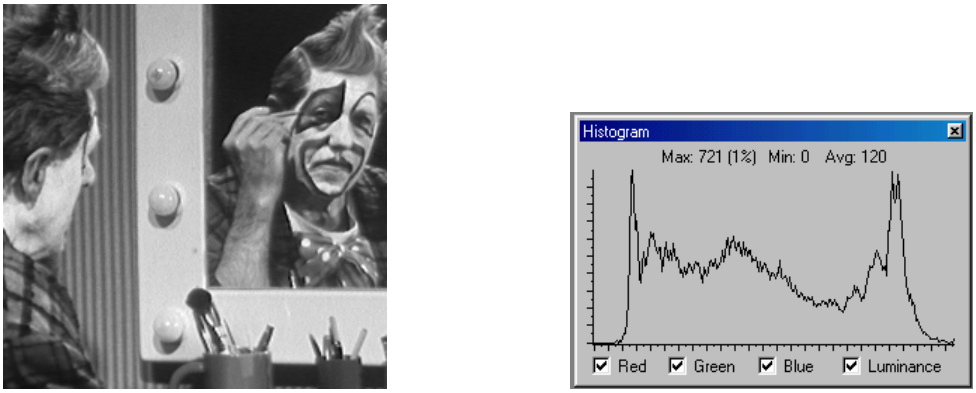


Figura 5–65 Imagen original ‘payaso.bmp’ y su histograma

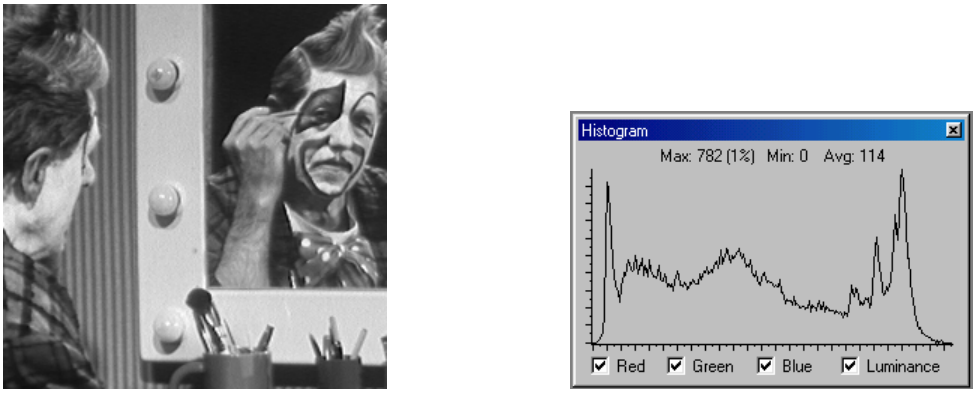


Figura 5–66 Imagen filtrada paso bajo (3 × 3) ‘payaso.bmp’ y su histograma

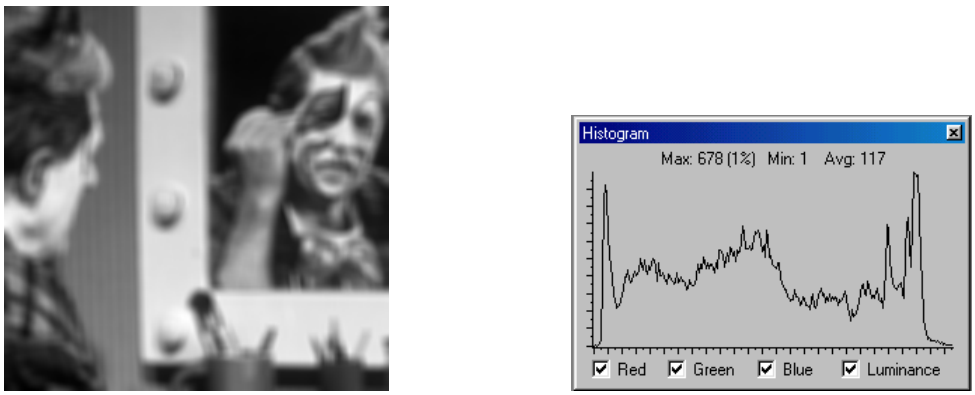


Figura 5–67 Imagen filtrada paso bajo (7 × 7) ‘payaso.bmp’ y su histograma

5.4.5 Filtro paso alto

El filtrado paso alto [2, 3, 4]se puede clasificar dentro de los filtros realzantes. Trata de destacar los detalles finos de una imagen o intensificar detalles que han sido difuminados. Las distintas aplicaciones donde existe son diversas pudiendo destacar la impresión electrónica, las imágenes médicas, las inspecciones industriales, etc.

A diferencia del filtro paso bajo para implementar un filtro espacial de paso alto se deben de tener coeficientes positivos cerca de su centro y coeficientes negativos en la periferia (todo

esto relativo a la máscara de convolución). Hay que tener en cuenta que la suma total de los coeficientes de la matriz de convolución debe de ser cero. Así, cuando la máscara está sobre un área de nivel de gris constante o poco variable, la salida proporcionada por la máscara es cero o un valor muy pequeño. Este resultado es coherente con lo que se espera del correspondiente filtro en el dominio de frecuencias.

Seguidamente se muestra un ejemplo mediante una máscara 3×3 donde en el centro hay un valor positivo y en el resto los coeficientes son negativos.

-1	-1	-1
-1	8	-1
-1	-1	-1

Tabla 5–6 Filtro espacial paso alto básico

La propiedad de que la suma total sea cero implica que la imagen puede tomar valores de gris negativos. Debido a que sólo consideramos niveles positivos de gris, los resultados obtenidos deben ser adaptados al intervalo [*0, 255*]. Una posible solución para no obtener valores negativos sería tomar el valor absoluto de los niveles de la imagen filtrada para que así todos los valores sean positivos, el problema es que los valores negativos grandes aparecerían muy resaltados en la imagen.

Al igual que antes la función que implementa la aplicación de una matriz de convolución se presenta en el apartado correspondiente a la aplicación de matrices de convolución. No obstante, se muestra el procedimiento que permite calcular la matriz de convolución paso alto.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Highpass(const ElementT number).

  Calcula la matriz de convolución a partir de un número de entrada de manera que todos los coeficientes de la matriz cumplan las restricciones establecidas. Se debe determinar el tamaño de la matriz antes de referenciar al procedimiento.

- ***Parámetros de entrada***

  number número de entrada.

- ***Parámetros de salida***

  matriz de salida.

Ejemplo

```
C_Matrix::IndexT sizeConv=3;
C_Matrix::ElementT number=1;

C_Trace ("Generatting The Convolution Matrix of High Pass...");
C_Matrix conv(1,sizeConv,1,sizeConv);
conv.Highpass(number);

C_Trace ("Writting The Convolution Matrix of High Pass...");
conv.Write(OutFile);

conv.Free();
```

Este ejemplo se limita a calcular y almacenar una matriz de convolución paso alto a partir de un parámetro de entrada.

Resultados

La matriz de convolución aplicada para mostrar los siguientes resultados es:

-1	-1	-1
-1	8	-1
-1	-1	-1

3×3

Laplaciano

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	25	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

5×5

Tabla 5–7 Matrices de convolución paso alto.

Se ha tomado esta matriz de convolución de dimensiones 5×5 al igual que antes para que el contraste sea mayor a la hora de comparar la imagen original con la filtrada.

En los resultados que se muestran a continuación se observan como trata de realzar los contornos a costa de realizar un suavizado en el centro de los objetos. Este realce de contornos se observa mejor en la figura del payaso que en el avión, esto puede deberse a que la matriz de convolución deba ser menor, así como, por la propia definición de la función del histograma que no permita una mejor extracción de los mismos.

Los resultados obtenidos para las distintas imágenes son los siguientes:

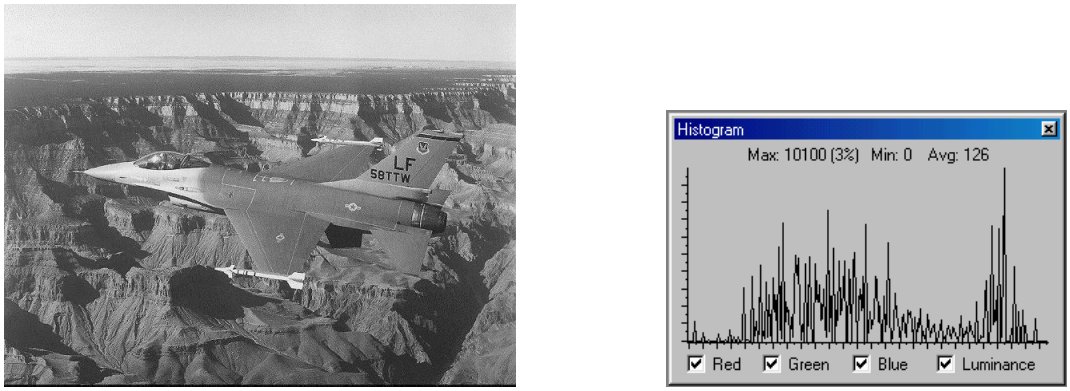


Figura 5–68 Imagen original ‘aviongr.bmp’ y su histograma

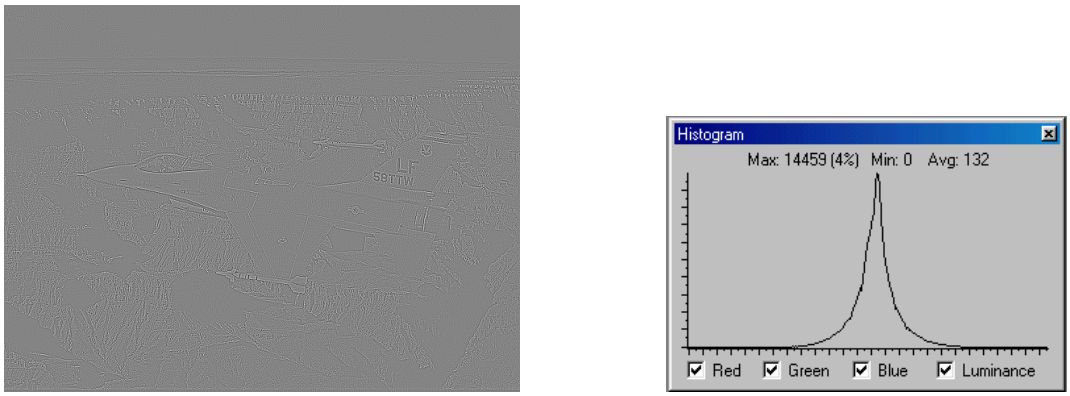


Figura 5–69 Imagen filtrada paso alto (3 × 3) ‘aviongr.bmp’ y su histograma

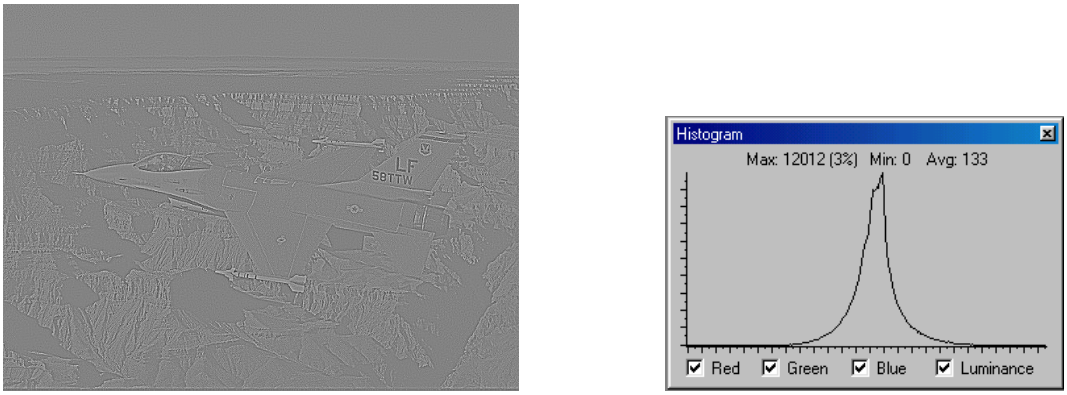


Figura 5–70 Imagen filtrada paso alto (5 × 5) ‘aviongr.bmp’ y su histograma

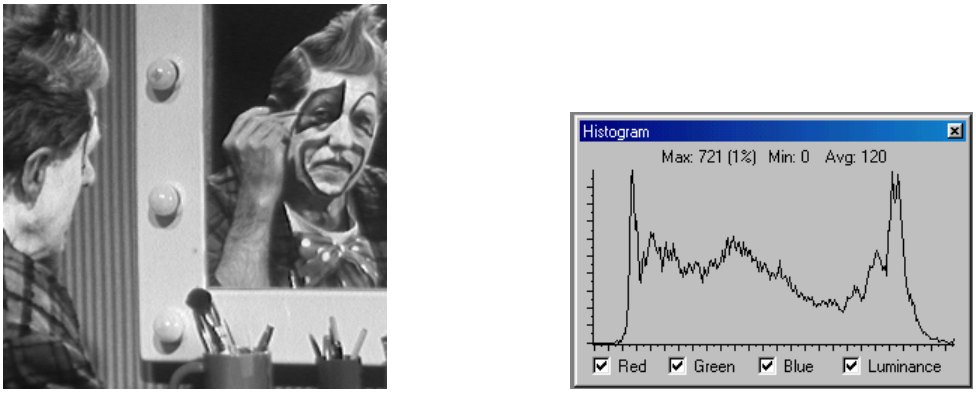


Figura 5–71 Imagen original ‘payaso.bmp’ y su histograma

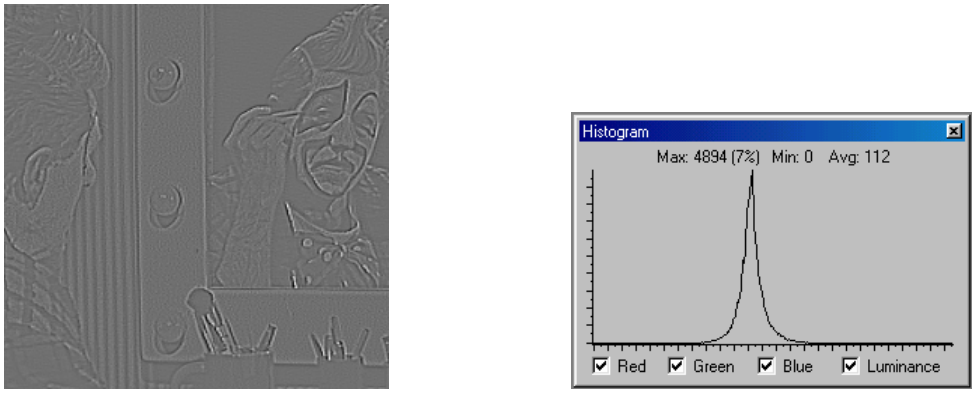


Figura 5–72 Imagen filtrada paso alto (3 × 3) ‘payaso.bmp’ y su histograma

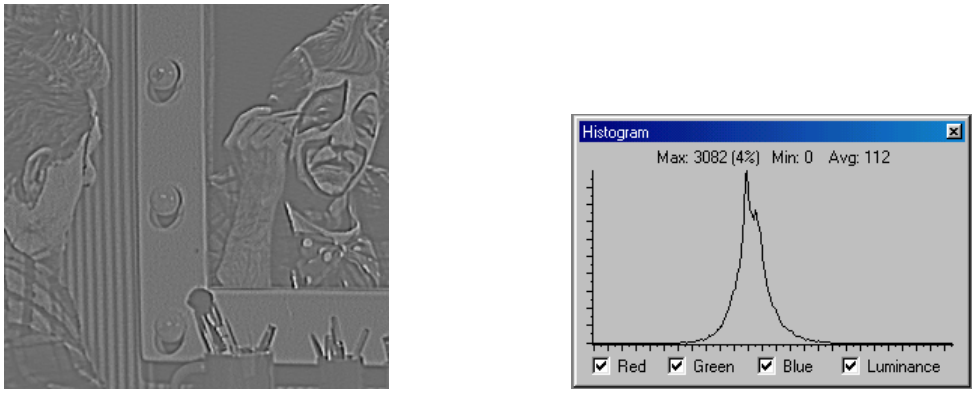


Figura 5–73 Imagen filtrada paso alto (5 × 5) ‘payaso.bmp’ y su histograma

Resultados de aplicación de matrices estándar

-0.5	-0.5	-0.5
-0.5	4.5	-0.5
-0.5	-0.5	-0.5

Tabla 5–8 Operador de forma (*sharpen*) 3 × 3

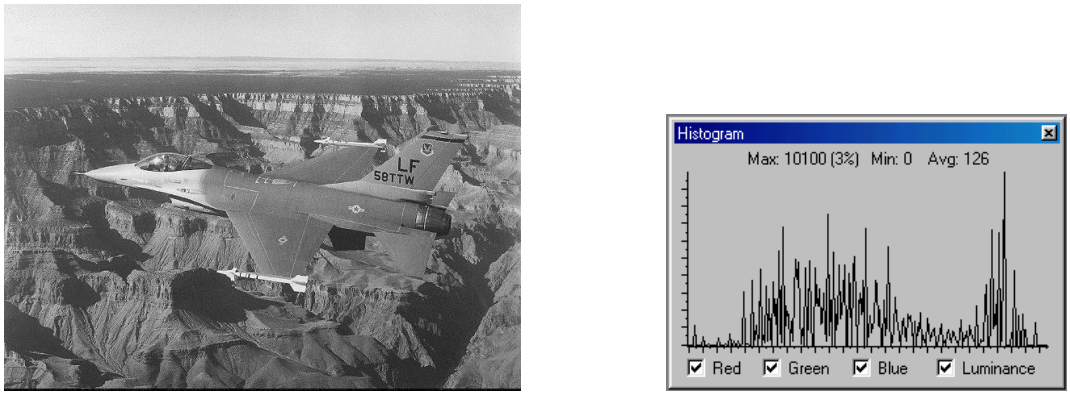


Figura 5–74 Imagen original ‘aviongr.bmp’ y su histograma

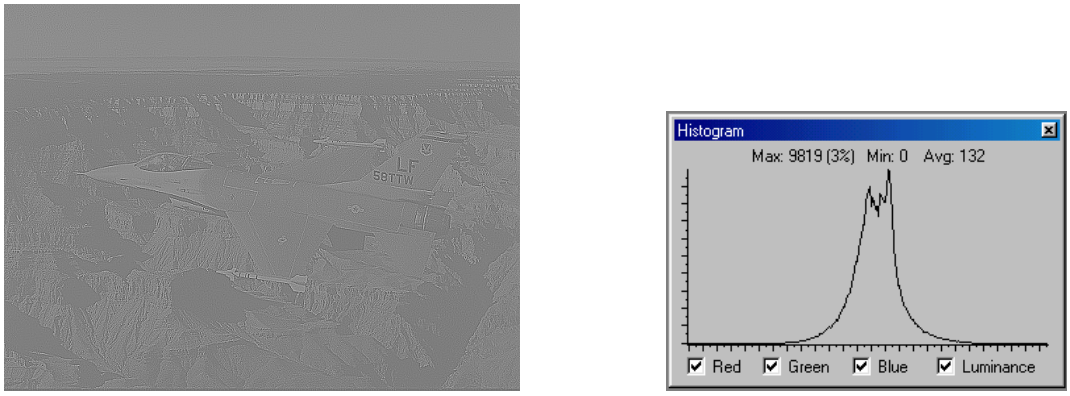


Figura 5–75 Imagen filtrada por el operador de forma ‘aviongr.bmp’ y su histograma

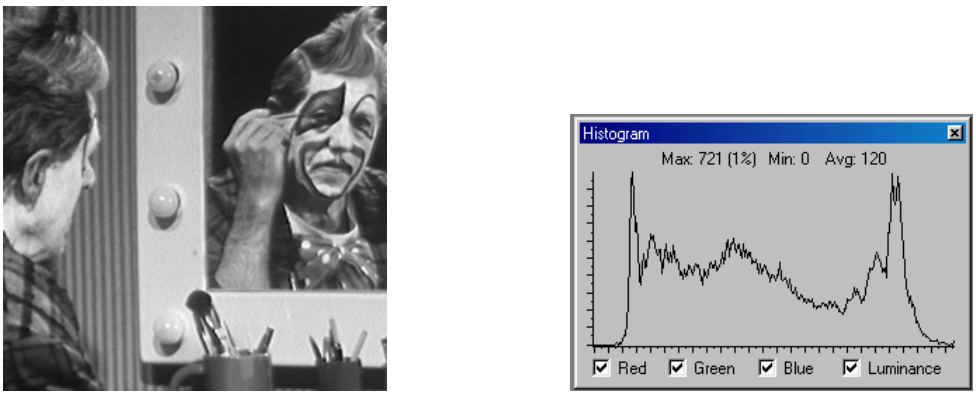


Figura 5–76 Imagen original ‘payaso.bmp’ y su histograma

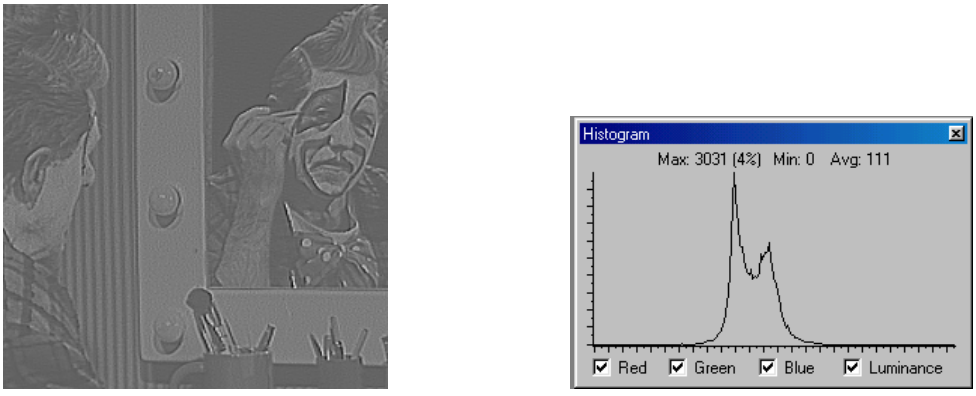


Figura 5–77 Imagen filtrada por el operador de forma ‘payaso.bmp’ y su histograma

Se observa para este operador como trata de extraer la forma de los objetos sin importar el relleno de las regiones que los contienen, es decir, se limita a la búsqueda de cambios de contraste para determinar bordes de regiones.

-1	-1	-1
-1	9	-1
-1	-1	-1

Tabla 5–9 Operador de forma 3 × 3

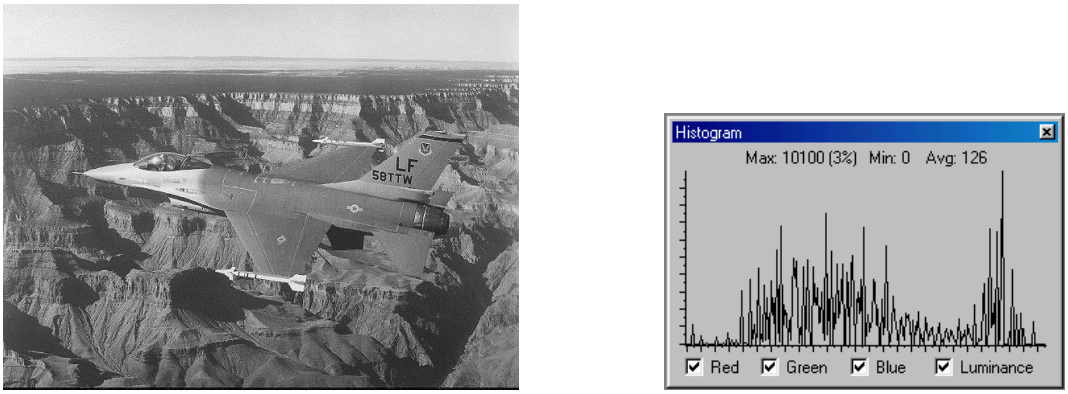


Figura 5–78 Imagen original ‘aviongr.bmp’ y su histograma

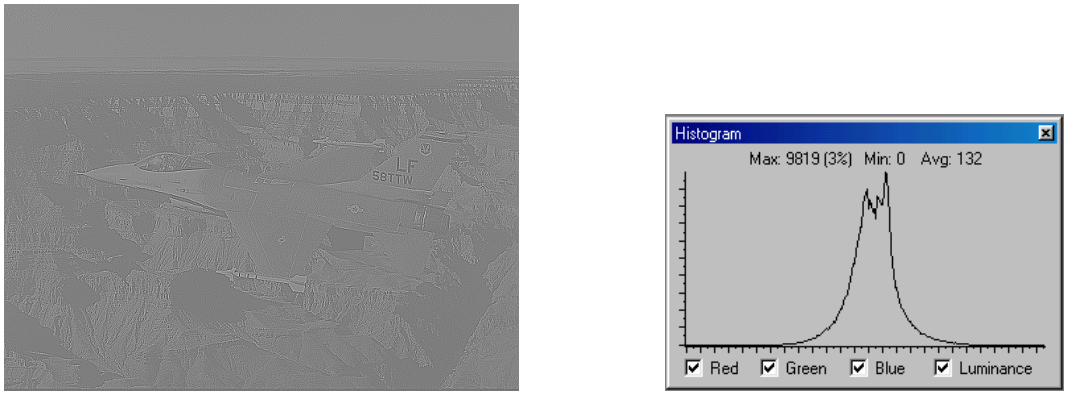


Figura 5–79 Imagen filtrada por el operador de forma ‘aviongr.bmp’ y su histograma

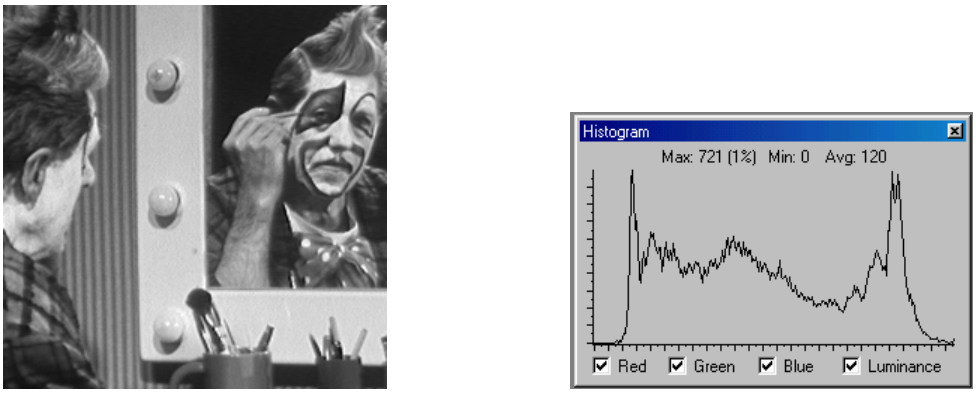


Figura 5–80 Imagen original ‘payaso.bmp’ y su histograma

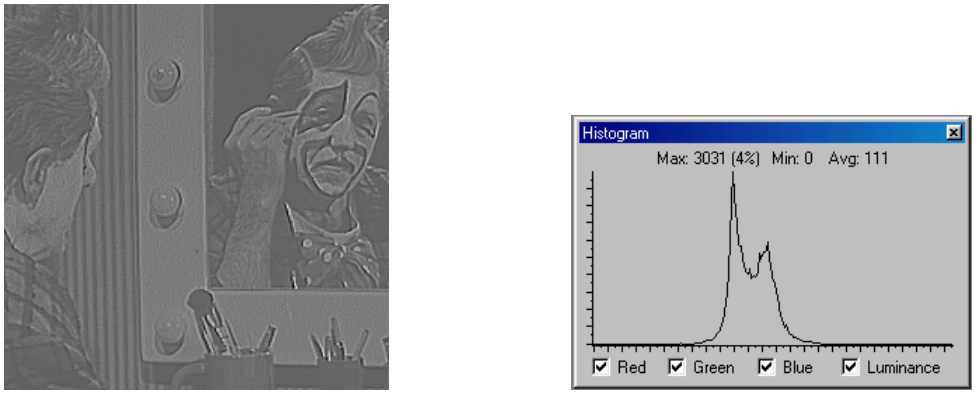


Figura 5–81 Imagen filtrada por el operador de forma 'payaso.bmp' y su histograma

5.4.6 Filtro de High-Boost

En cierta medida, al igual que el filtro paso alta el High-Boost [2] es considerado como un filtro realzante, ya que trata de destacar los detalles finos de una imagen. Una imagen filtrada de paso alto puede ser calculada como la diferencia entre la imagen original y una versión de esta imagen que ha pasado por un filtro de paso bajo, es decir:

$$\text{Paso alto} = \text{Original} - \text{Paso bajo}$$

Ecuación 5–18 Cálculo del paso alto

Multiplicando la imagen original por un factor de amplificación, al que se le ha denotado A, se obtiene la ecuación para el filtro *High-Boost*:

$$\begin{aligned}\text{High-Boost} &= (A)(\text{Original}) - \text{Paso bajo} = \\ &= (A-1)(\text{Original}) + \text{Original} - \text{Paso bajo} = \\ &= (A-1)(\text{Original}) + \text{Paso alto}\end{aligned}$$

Ecuación 5–19 Fórmula de High-Boost

Dependiendo del valor que tome A podemos obtener los siguientes resultados:

- A = 1: da el resultado de un filtro de paso alto normal.
- A > 1: parte del propio original se añade al resultado del filtro de paso alto, lo que devuelve parcialmente las componentes de frecuencias bajas pérdidas en el proceso del filtrado de paso alto. El resultado es que la imagen *High-Boost* se parece más a la imagen original, con un grado relativo de mejora de los bordes que dependen del valor de *A*.

El proceso general de sustraer una imagen difusa de una original se denomina enmascaramiento difuminado. Estos filtros se usan en las aplicaciones para el procesamiento de imágenes en las industrias de artes gráficas.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  HighBoost(C_Matrix &conv,const ElementT A).

  Calcula el filtro de HighBoost a partir del coeficiente multiplicativo de la matriz.

- ***Parámetros de entrada***

  conv matriz de convolución paso bajo.

  A coeficiente multiplicativo de la matriz.

- ***Parámetros de salida***

  matriz de salida.

El procedimiento se efectúa en dos pasos:

- Cálculo de la imagen filtrada a partir de la matriz de convolución aplicada.
- Aplicar la ecuación definida para el filtro de *High-Boost*.

$$\text{High-Boost} = (A - 1)(\text{Original}) + \text{Paso alto}$$

Ecuación 5–20 Fórmula de High-Boost

Ejemplo

```
C_Matrix::IndexT sizeConv=3;
C_Matrix::ElementT number=1,A=1;
C_Image image;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

C_Trace ("Generatting The Convolution Matrix of High Pass...");
C_Matrix conv(1,sizeConv,1,sizeConv);
conv.Highpass(number);

C_Trace ("Generatting The Convolution Matrix of High Boost...");
image.HighBoost(conv,A);

C_Matrix::IndexT rowN,colN,colorsN;
C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
image.Stretch(0,(colorsN-1));

C_Trace ("Writting High Boost Image...");
image.WriteBMP (OutFile);

conv.Free();
```

Resultados

Presentaremos los resultados obtenidos al aplicar el filtro de *High-Boost* para las dos imágenes cambiando el valor de A para ver el resultado producido por este factor. La matriz aplicada ha sido la correspondiente a la Figura 5–70 correspondiente al filtro paso alto.

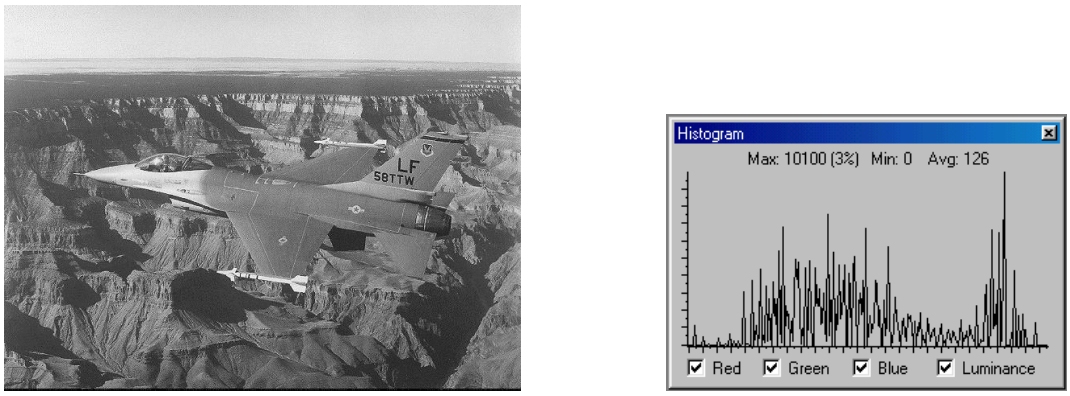


Figura 5–82 Imagen original 'aviongr.bmp' y su histograma

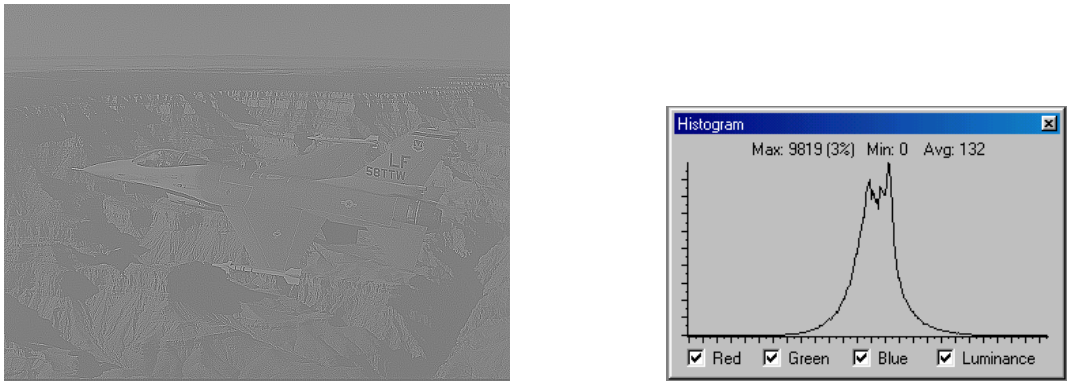


Figura 5–83 Imagen filtrada por High-Boost (A=1) 'aviongr.bmp' y su histograma

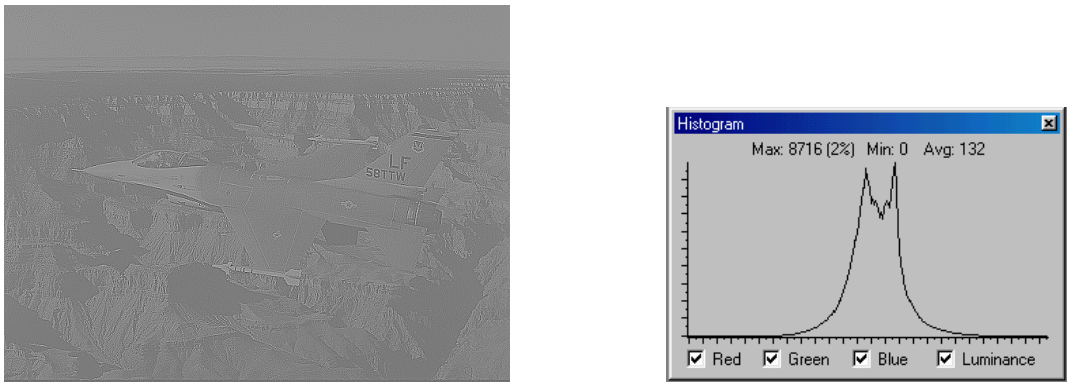


Figura 5–84 Imagen filtrada por High-Boost (A=1.5) 'aviongr.bmp' y su histograma

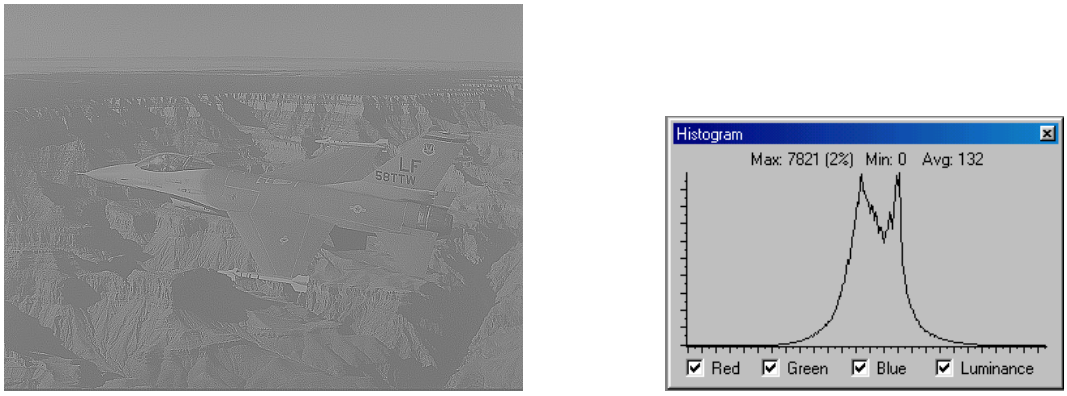


Figura 5–85 Imagen filtrada por High-Boost (A=2) 'aviongr.bmp' y su histograma

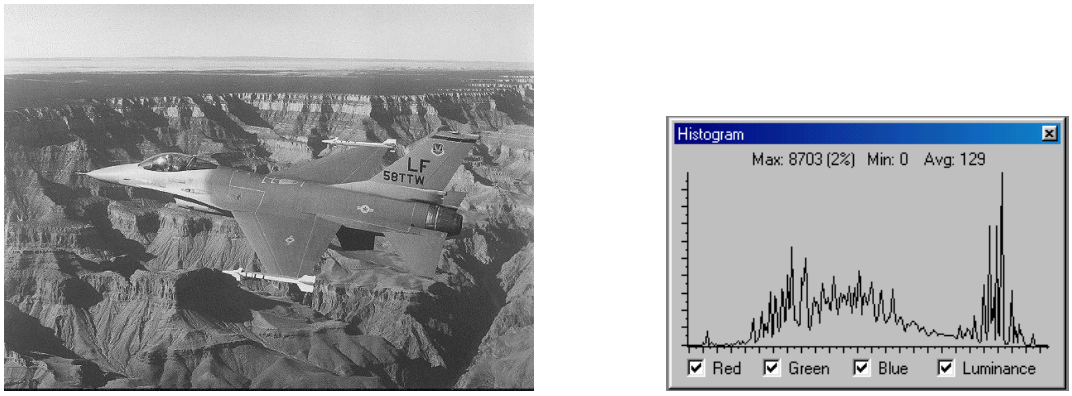


Figura 5–86 Imagen filtrada por High-Boost (A=100) 'aviongr.bmp' y su histograma

Podemos comprobar como conforme aumenta el valor de A hay una mayor aproximación a la imagen original. Además, el histograma de las diferentes imágenes es el que da una visión más general de lo que va sucediendo. En los primeros tres histogramas los niveles de gris se concentran en los valores centrales y en el último se obtiene una curva mejorada de la imagen en cuestión.

Para la imagen del payaso, el filtro de *High-Boost* produce los siguientes resultados:

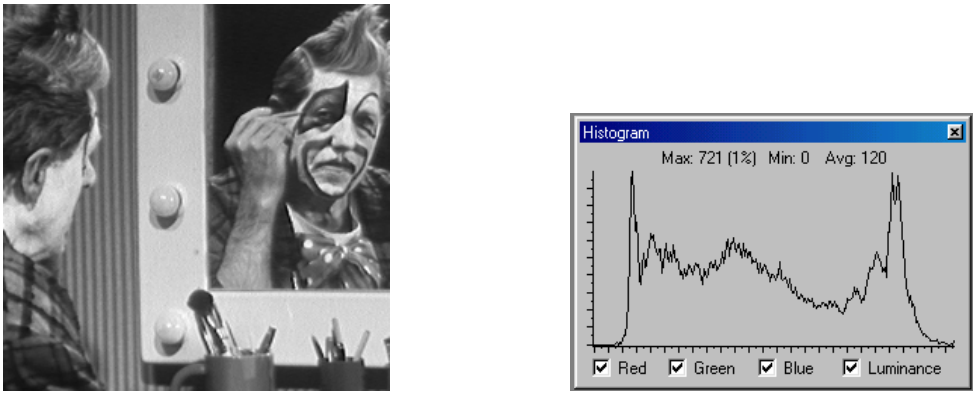


Figura 5–87 Imagen original 'payaso.bmp' y su histograma

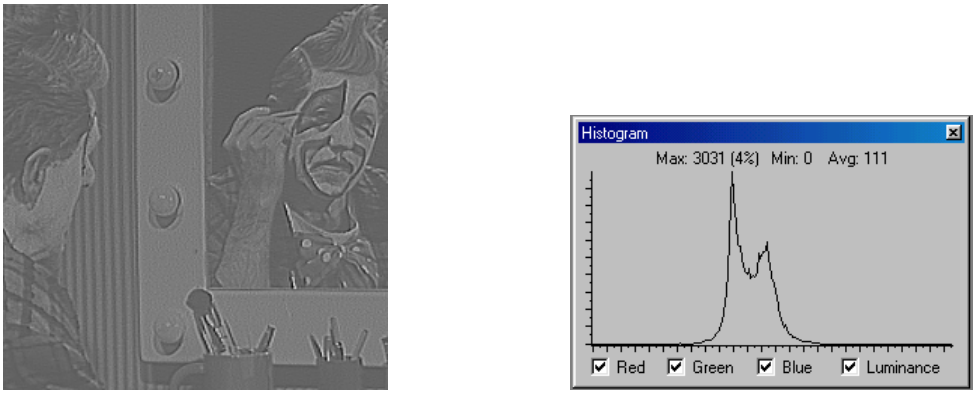


Figura 5–88 Imagen filtrada por High-Boost (A=1) ‘payaso.bmp’ y su histograma

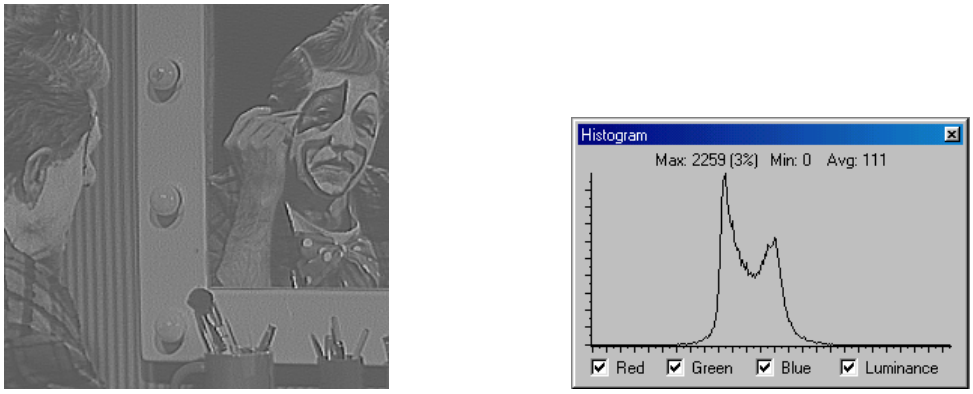


Figura 5–89 Imagen filtrada por High-Boost (A=1.5) ‘payaso.bmp’ y su histograma

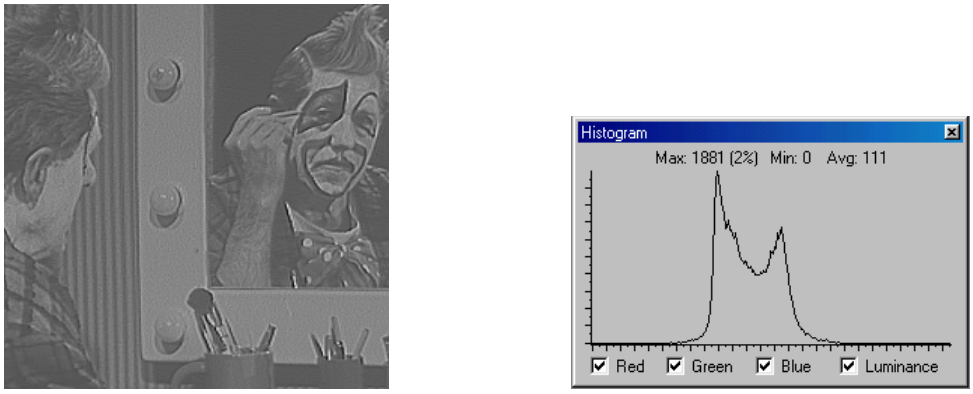


Figura 5–90 Imagen filtrada por High-Boost (A=2) ‘payaso.bmp’ y su histograma

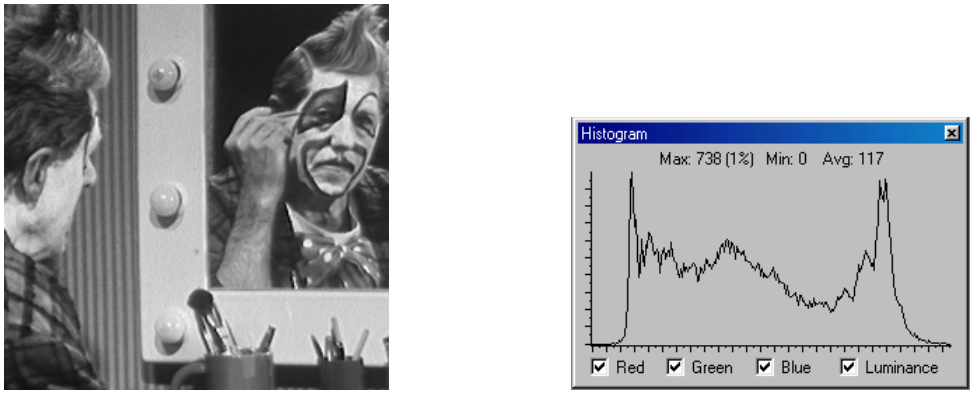


Figura 5–91 Imagen filtrada por High-Boost (A=2) 'payaso.bmp' y su histograma

Al igual que los comentarios realizados antes para la Figura 5–91se observa que para valores bajos de A se obtiene una imagen suave de niveles de gris donde se describen las formas de los distintos objetos. Mientras que a partir de un valor alto de A, la imagen tiende a semejarse a la imagen de partida.

5.4.7 Filtro de Gauss.

El operador del filtro Gaussiano [2, 5] es un operador de convolución para imágenes que se utiliza para emborronar imágenes y eliminar detalles y ruido. En este sentido es similar al filtro de la mediana, pero este filtro utiliza una matriz de convolución distinta que representa la forma de la función de una función Gaussiano (en forma de campana). Esta matriz presenta algunas propiedades especiales.

La distribución Gaussiano en una sola dimensión tiene la forma:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

Ecuación 5–21 Función de distribución de Gauss unidimensional

donde σ es la desviación estándar de la distribución. Tenemos que suponer que la media de la distribución es cero.

La distribución de la función Gaussiano se representa en la siguiente figura:

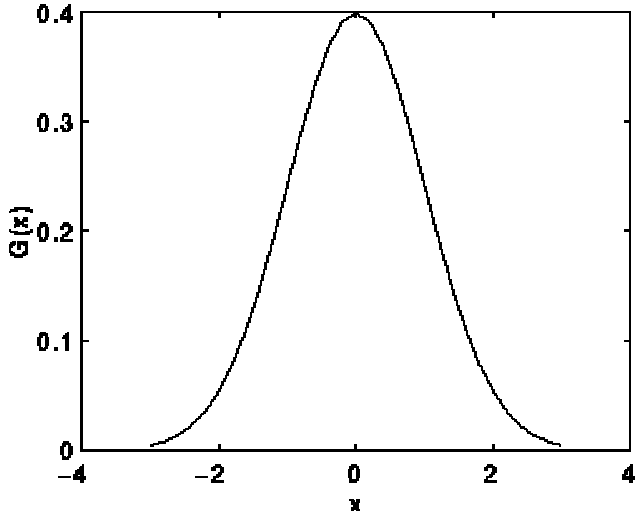


Figura 5–92 Distribución Gaussiano en 1-D

En dos dimensiones, una distribución Gaussiano presenta la forma:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Ecuación 5–22 Función de distribución de Gauss bidimensional

Cuya representación se muestra en la siguiente figura:

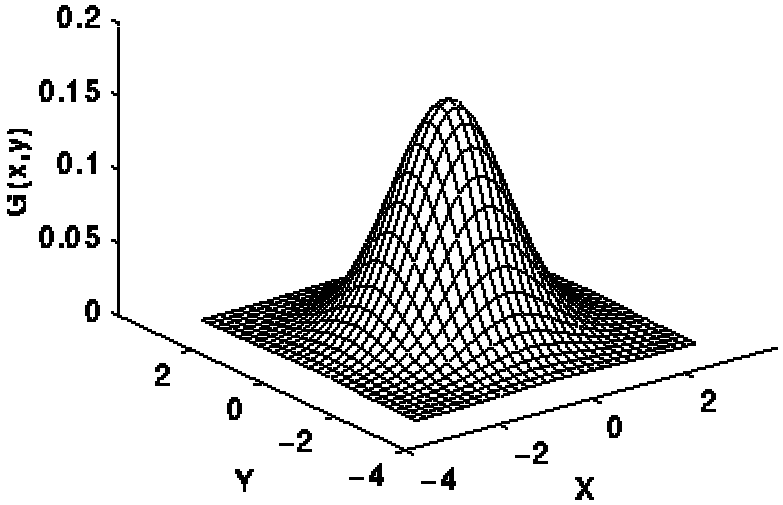


Figura 5–93 Distribución Gaussiano en 2-D

La idea del filtro Gaussiano es utilizada en su forma 2-D como una función de expansión del punto. Dado que la imagen se almacena como una colección de puntos discretos, necesitamos producir una aproximación discreta a la función Gaussiano antes de poder aplicar la convolución. Se puede calcular una matriz que simule la distribución Gaussiano, entonces el filtro puede simularse utilizando métodos de convolución estándar.

Una forma adelantada de implementar el filtro Gaussiano es con una gran desviación estándar, por lo que resulta más difícil obtener resultados en lugar de aplicar pequeños filtros gaussianos en varias veces a la imagen original. Mientras que esto es computacionalmente complejo, puede tener aplicabilidad si el procesamiento se esta llevando a cabo utilizando un hardware segmentado.

El procedimiento implementado recibe como parámetro el valor de la desviación estándar que va a ser utilizado en el cálculo de la matriz de convolución. Posteriormente calcula el valor de x e y de la posición a tratar y se le asigna al píxel correspondiente de salida el valor que resulta al calcular la ecuación definida para la Distribución Gaussiano.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  Gaussian (const float std).

  Calcula la matriz del Gaussiano para que posteriormente puede ser aplicada como una matriz de convolución más a la imagen.

- ***Parámetros de entrada***

  std desviación estándar de la fórmula para el cálculo.

- ***Parámetros de salida***

  matriz de convolución de salida.

El procedimiento explicado solo devuelve la matriz de convolución lista para ser aplicada. En el apartado correspondiente a la aplicación de matrices de convolución. se describen los distintos métodos especificados para la aplicación de matrices de convolución.

Ejemplo

```
C_Matrix::IndexT sizeConv=3;
C_Matrix::ElementT desviation;

C_Trace ("Generatting The Convolution Matrix of Gaussian...");
C_Matrix conv(1,sizeConv,1,sizeConv);
conv.Gaussian(desviation);

C_Trace ("Writting The Convolution Matrix of Gaussian...");
conv.Write(OutFile);

conv.Free();
```

Este ejemplo se limita a calcular y almacenar una matriz de convolución paso alto a partir de un parámetro de entrada.

Resultados

La matriz de convolución usada para el cálculo de los distintos resultados es la siguiente:

0.0283853	0.0362151	0.0392785	0.0362151	0.0283853
0.0362151	0.0462047	0.0501131	0.0462047	0.0362151
0.0392785	0.0501131	0.0543522	0.0501131	0.0392785
0.0362151	0.0462047	0.0501131	0.0462047	0.0362151
0.0283853	0.0362151	0.0392785	0.0362151	0.0283853

Tabla 5–10 Matriz de convolución con los parámetros desviación = 1.4 y tamaño = 5 × 5

En los resultados obtenidos de aplicar la matriz anterior se contempla como se produce un suavizado en los contornos de las imágenes dando una sensación de borrosidad. Donde se ve más claramente esto es al comparar los histogramas de la imagen original y la resultante del filtro. Se observa como tiende a eliminar los picos redondeando la función del histograma.

Los resultados obtenidos para las distintas imágenes son los siguientes:

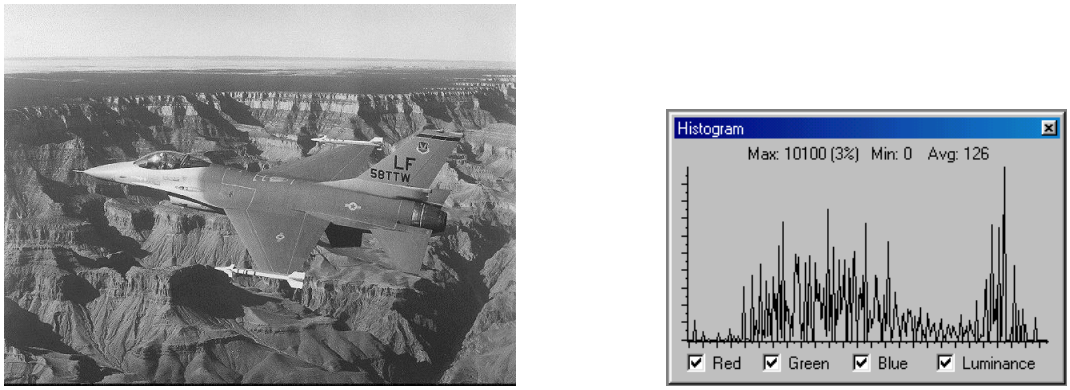


Figura 5–94 Imagen original ‘aviongr.bmp’ y su histograma

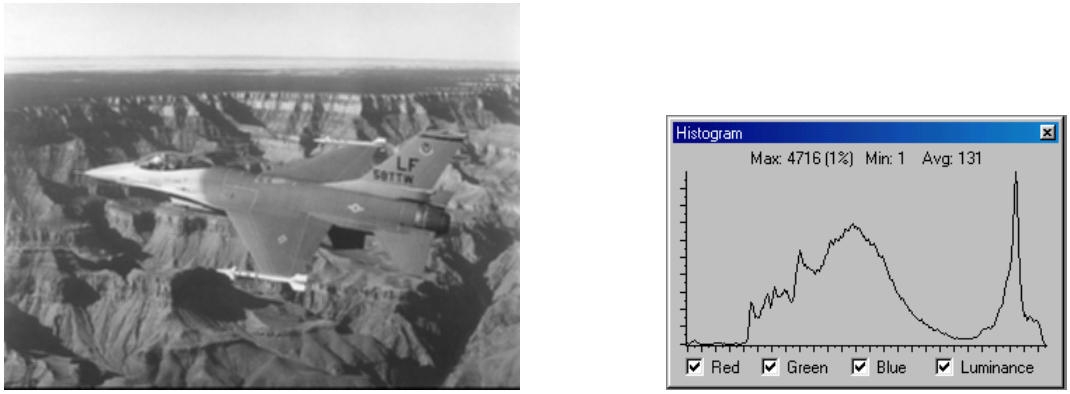


Figura 5–95 Imagen filtrada por la gaussiana (5 × 5, d=1.4) ‘aviongr.bmp’ y su histograma

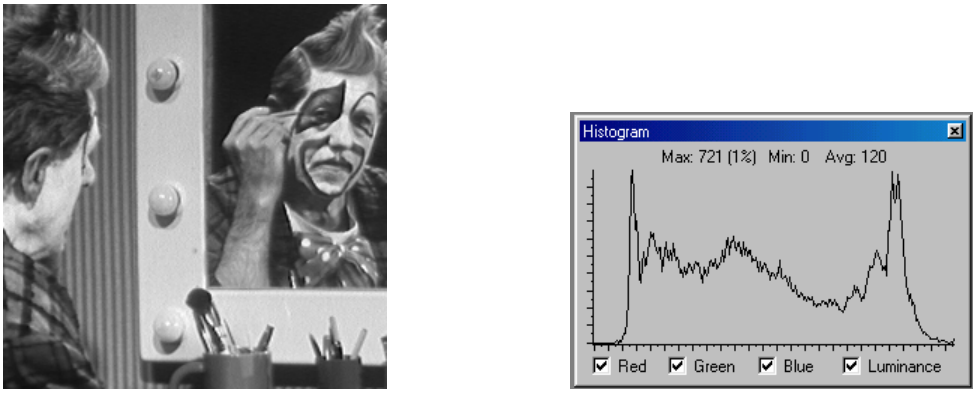


Figura 5–96 Imagen original ‘payaso.bmp’ y su histograma

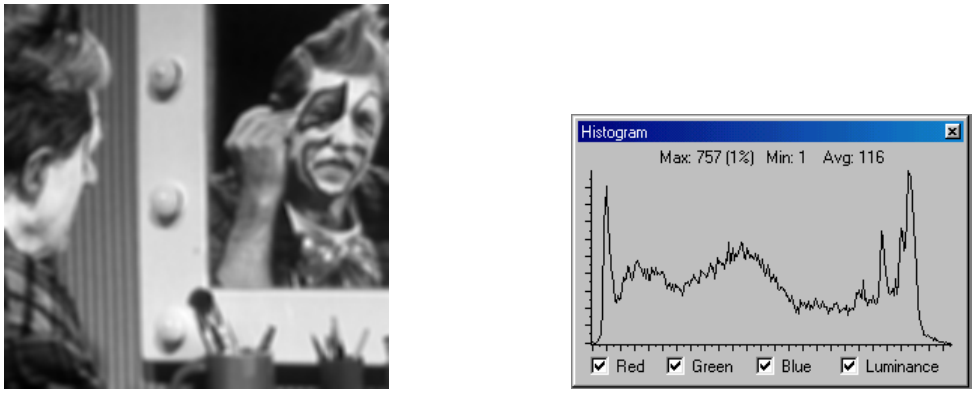


Figura 5–97 Imagen filtrada por la gaussiana (5 × 5, d=1.4) ‘payaso.bmp’ y su histograma

Resultados de aplicación de matrices estándar

0.0625	0.125	0.0625
0.125	0.250	0.125
0.0625	0.125	0.0625

Tabla 5–11 Operador de la gaussiana 3 × 3

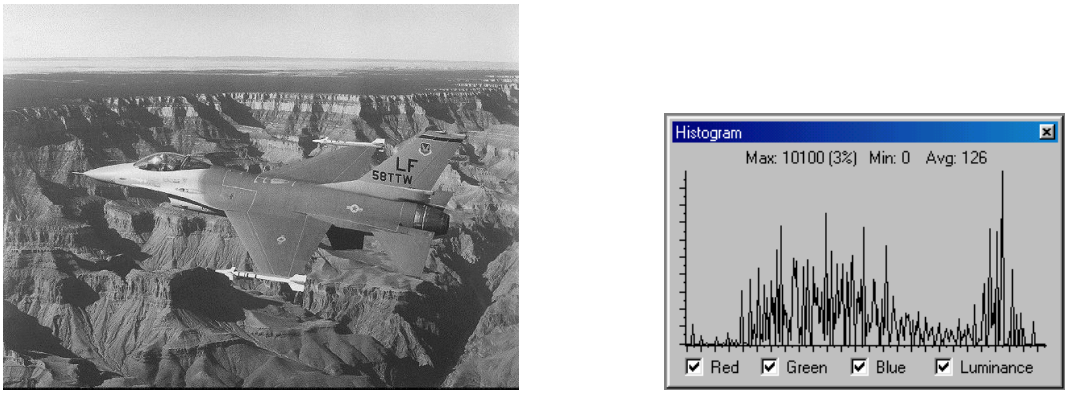


Figura 5–98 Imagen original ‘aviongr.bmp’ y su histograma

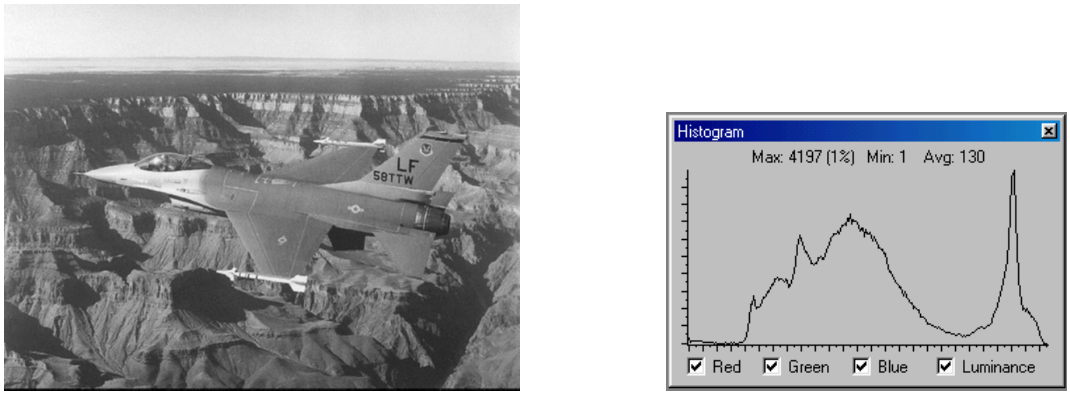


Figura 5–99 Imagen filtrada por el operador de la gaussiana ‘aviongr.bmp’ y su histograma

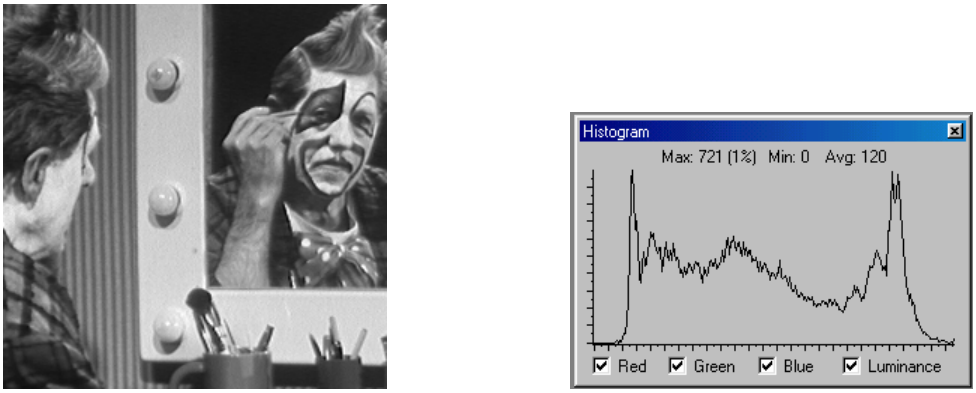


Figura 5–100 Imagen original ‘payaso.bmp’ y su histograma

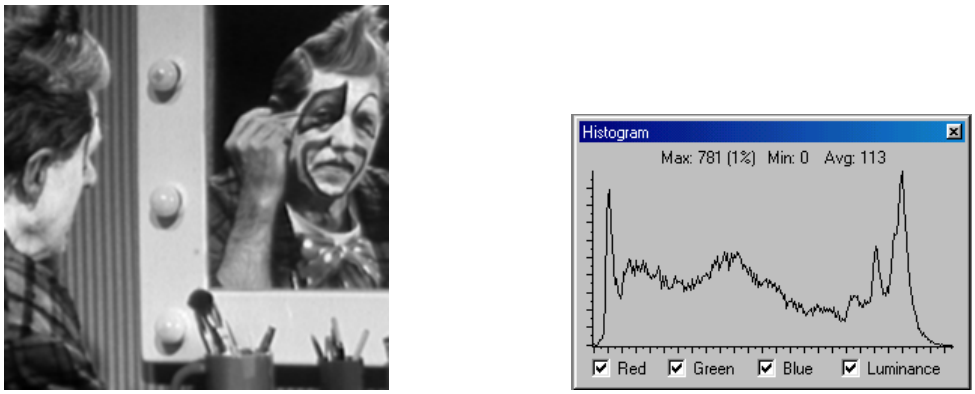


Figura 5–101 Imagen filtrada por el operador de la gaussiana ‘payaso.bmp’ y su histograma

0	1	2	1	0
1	2	4	2	1
2	4	9	4	2
1	2	4	2	1
0	1	2	1	0

Tabla 5–12 Operador de la gaussiana 5 × 5

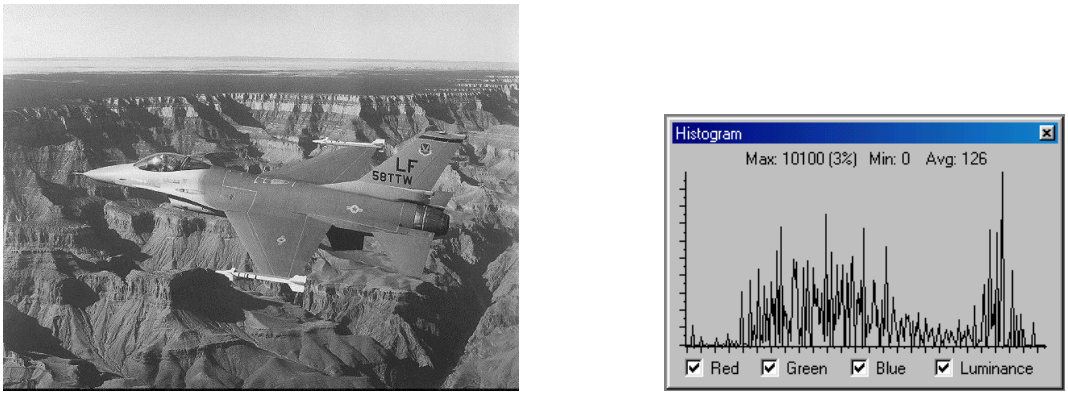


Figura 5–102 Imagen original ‘aviongr.bmp’ y su histograma

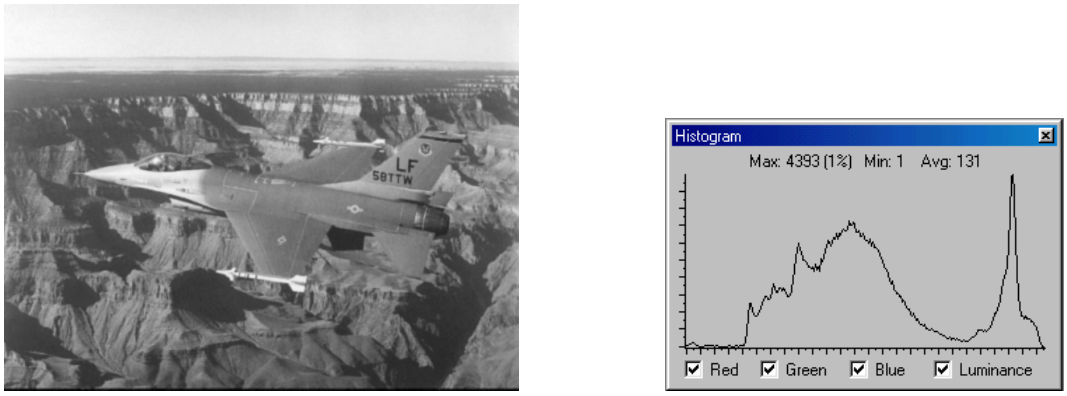


Figura 5–103 Imagen filtrada por el operador de la gaussiana ‘aviongr.bmp’ y su histograma

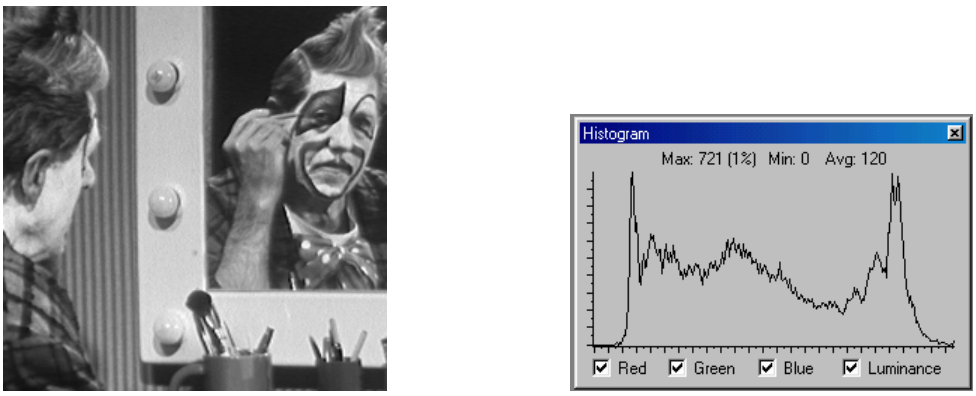


Figura 5–104 Imagen original ‘payaso.bmp’ y su histograma

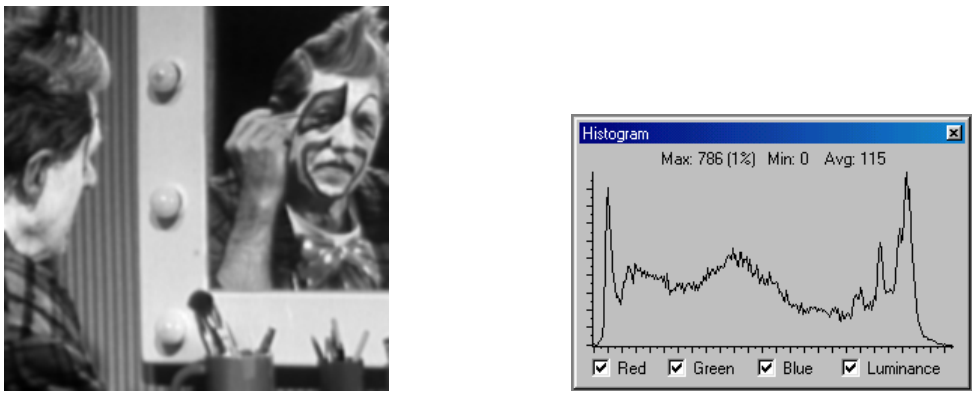


Figura 5–105 Imagen filtrada por el operador de la gaussiana ‘payaso.bmp’ y su histograma

0	0	1	1	1	0	0
0	2	8	13	8	2	0
1	8	36	60	36	8	1
1	13	60	99	60	13	1
1	8	36	60	36	8	1
0	2	8	13	8	2	0
0	0	1	1	1	0	0

Tabla 5–13 Operador de la gaussiana 7 × 7

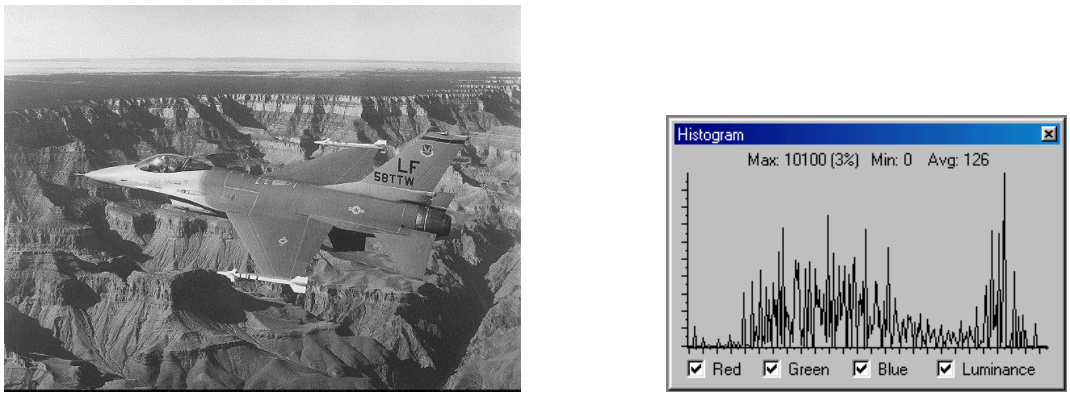


Figura 5–106 Imagen original ‘aviongr.bmp’ y su histograma

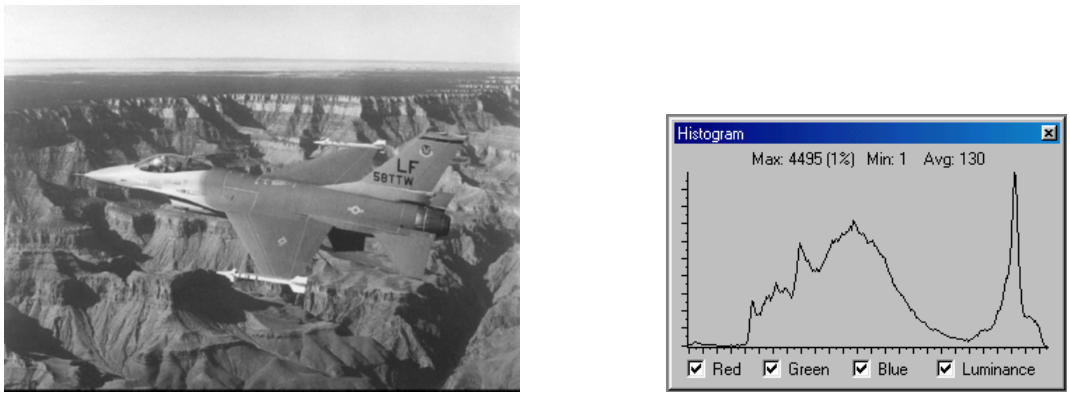


Figura 5–107 Imagen filtrada por el operador de la gaussiana ‘aviongr.bmp’ y su histograma

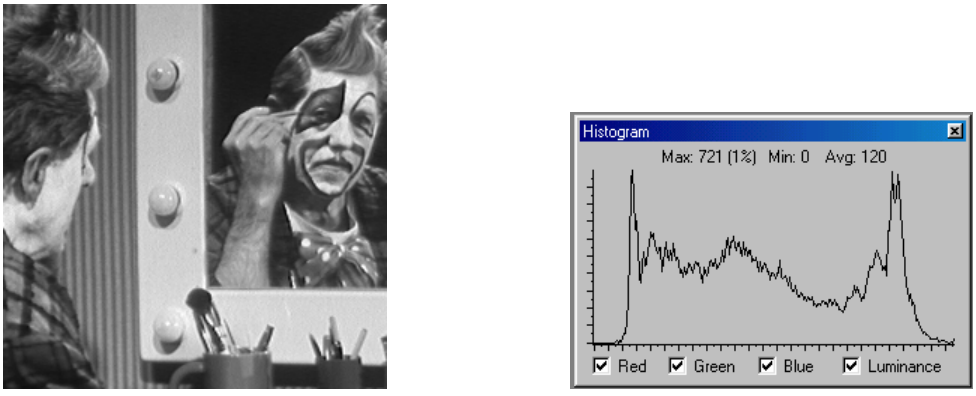


Figura 5–108 Imagen original ‘payaso.bmp’ y su histograma

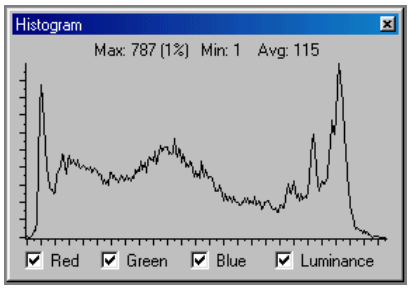


Figura 5–109 Imagen filtrada por el operador de la gaussiana ‘payaso.bmp’ y su histograma

0	0	1	1	2	1	1	0	0
0	1	4	8	10	8	4	1	0
1	4	13	28	36	28	13	4	1
1	8	28	60	77	60	28	8	1
2	10	36	77	99	77	36	10	2
1	8	28	60	77	60	28	8	1
1	4	13	28	36	28	13	4	1
0	1	4	8	10	8	4	1	0
0	0	1	1	2	1	1	0	0

Tabla 5–14 Operador de la gaussiana 9 × 9

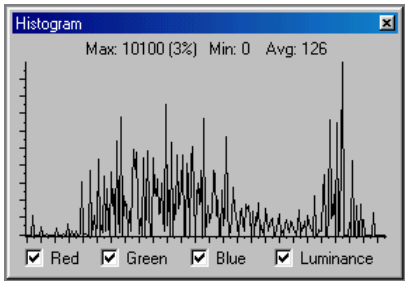


Figura 5–110 Imagen original ‘aviongr.bmp’ y su histograma

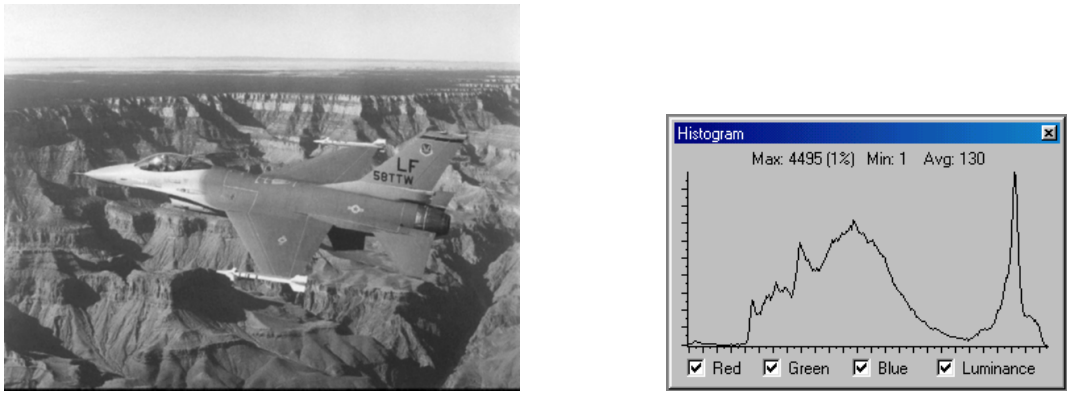


Figura 5–111 Imagen filtrada por el operador de la gaussiana ‘aviongr.bmp’ y su histograma

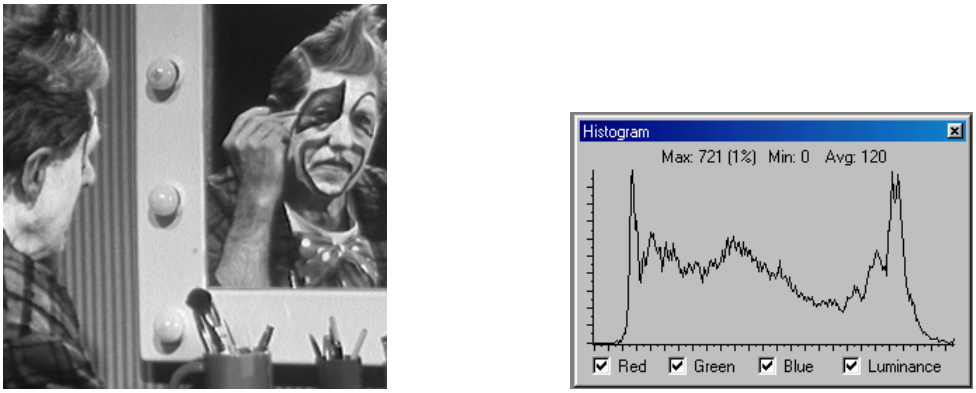


Figura 5–112 Imagen original ‘payaso.bmp’ y su histograma

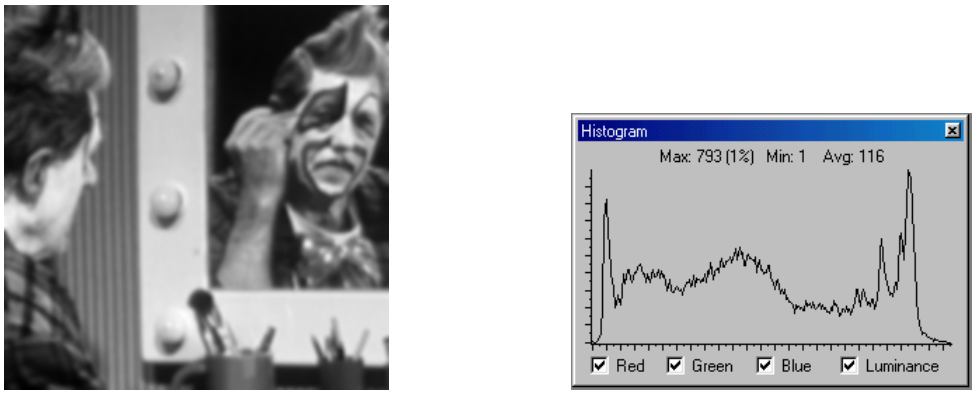


Figura 5–113 Imagen filtrada por el operador de la gaussiana ‘payaso.bmp’ y su histograma

0	0	0	0	1	1	1	1	1	0	0	0	0
0	0	1	1	3	4	4	4	3	1	1	0	0
0	1	2	4	8	12	13	12	8	4	2	1	0
0	1	4	10	19	28	32	28	19	10	4	1	0
1	3	8	19	36	53	60	53	36	19	8	3	1
1	4	12	28	53	77	87	77	53	28	12	4	1
1	4	13	32	60	87	99	87	60	32	13	4	1
1	4	12	28	53	77	87	77	53	28	12	4	1
1	3	8	19	36	53	60	53	36	19	8	3	1
0	1	4	10	19	28	32	28	19	10	4	1	0
0	1	2	4	8	12	13	12	8	4	2	1	0
0	0	1	1	3	4	4	4	3	1	1	0	0
0	0	0	0	1	1	1	1	1	0	0	0	0

Tabla 5–15 Operador de la gaussiana 13 × 13

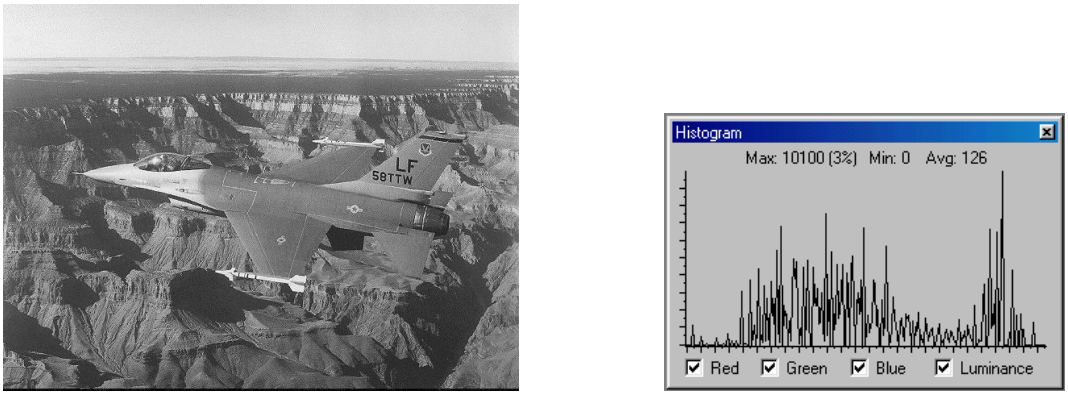


Figura 5–114 Imagen original ‘aviongr.bmp’ y su histograma

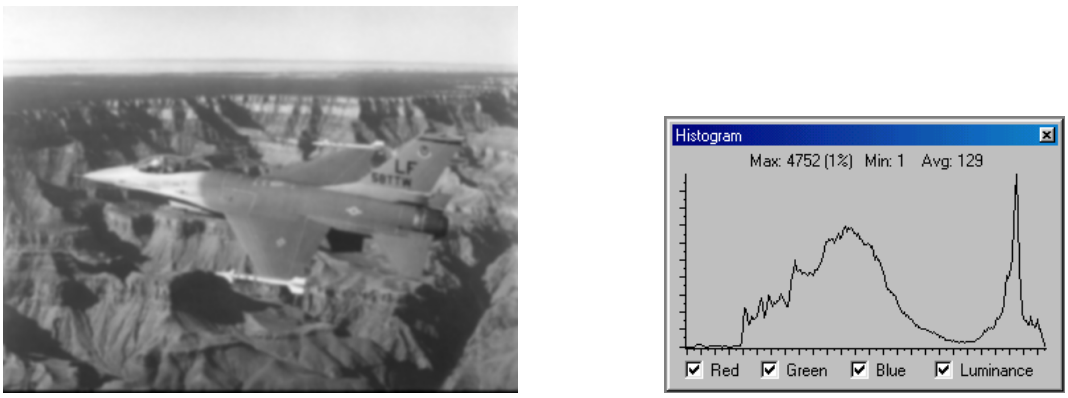


Figura 5–115 Imagen filtrada por el operador de la gaussiana ‘aviongr.bmp’ y su histograma

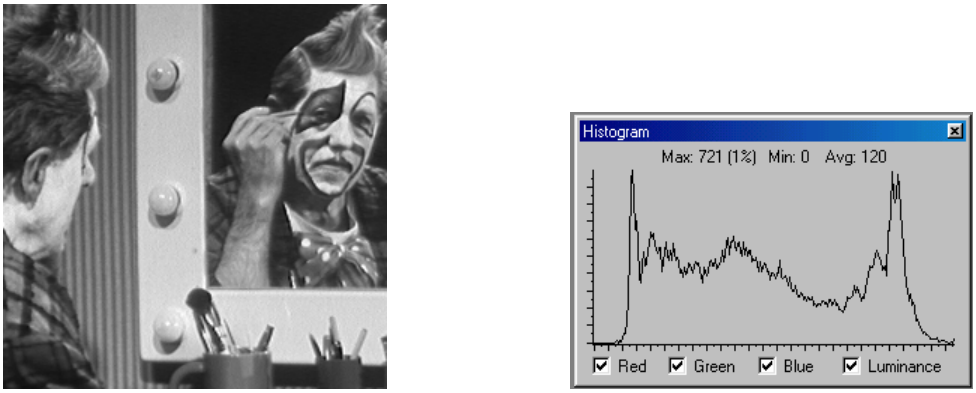


Figura 5–116 Imagen original ‘payaso.bmp’ y su histograma

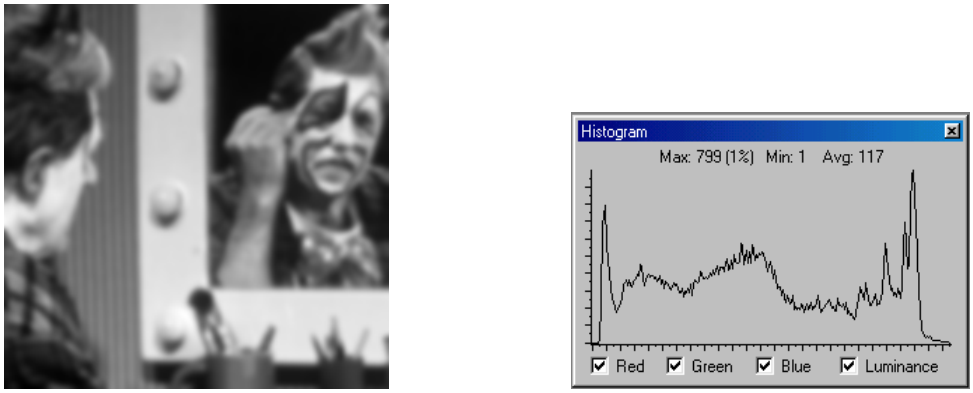


Figura 5–117 Imagen filtrada por el operador de la gaussiana ‘payaso.bmp’ y su histograma

0	0	1	2	3	4	4	5	4	4	3	2	1	0	0
0	1	2	4	6	8	10	10	10	8	6	4	2	1	0
1	2	4	8	12	16	19	21	19	16	12	8	4	2	1
2	4	8	13	21	28	34	36	34	28	21	13	8	4	2
3	6	12	21	32	44	53	56	53	44	32	21	12	6	3
4	8	16	28	44	60	72	77	72	60	44	28	16	8	4
4	10	19	34	53	72	87	93	87	72	53	34	19	10	4
5	10	21	36	56	77	93	99	93	77	56	36	21	10	5
4	10	19	34	53	72	87	93	87	72	53	34	19	10	4
4	8	16	28	44	60	72	77	72	60	44	28	16	8	4
3	6	12	21	32	44	53	56	53	44	32	21	12	6	3
2	4	8	13	21	28	34	36	34	28	21	13	8	4	2
1	2	4	8	12	16	19	21	19	16	12	8	4	2	1
0	1	2	4	6	8	10	10	10	8	6	4	2	1	0
0	0	1	2	3	4	4	5	4	4	3	2	1	0	0

Tabla 5–16 Operador de la gaussiana 15 × 15

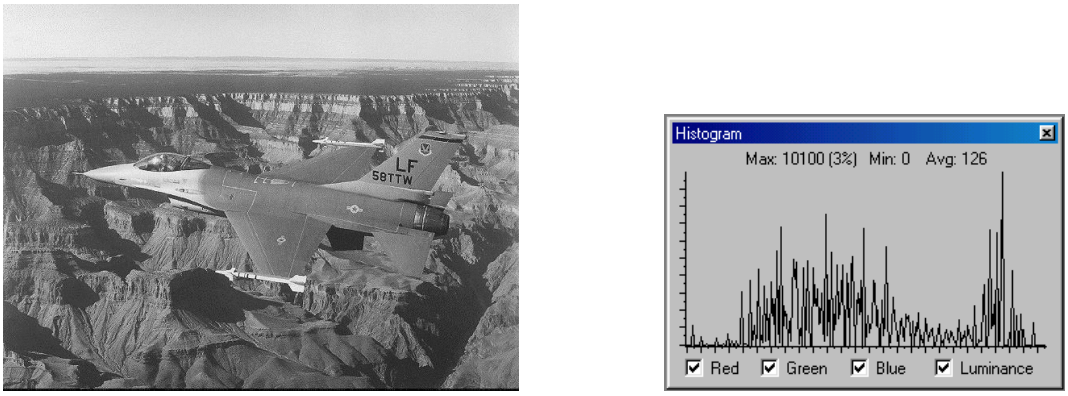


Figura 5–118 Imagen original ‘aviongr.bmp’ y su histograma

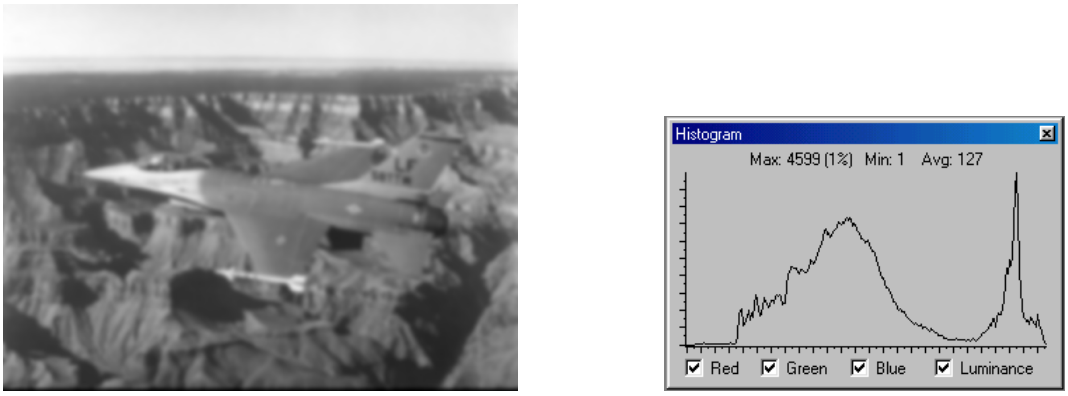


Figura 5–119 Imagen filtrada por el operador de la gaussiana ‘aviongr.bmp’ y su histograma

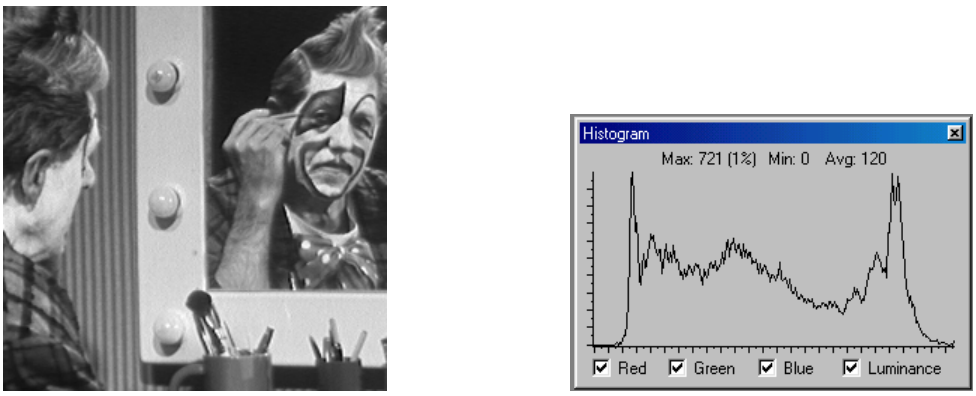


Figura 5–120 Imagen original ‘aviongr.bmp’ y su histograma

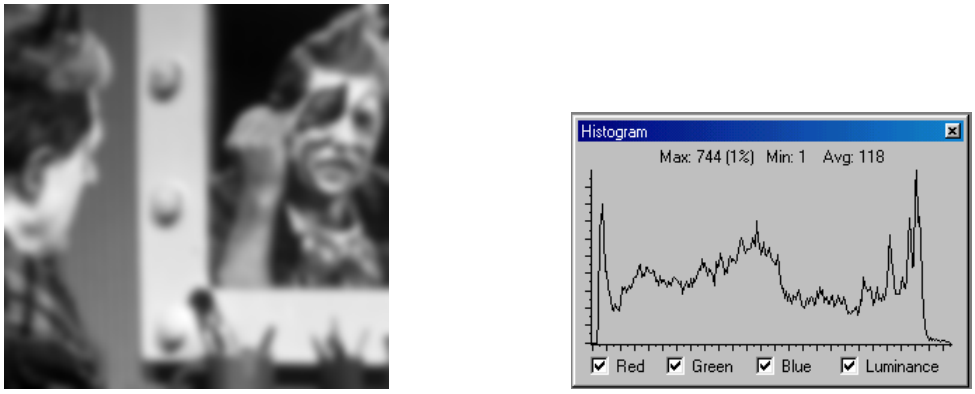


Figura 5–121 Imagen filtrada por el operador de la gaussiana ‘payaso.bmp’ y su histograma

Para cada uno de los ejemplos observados se puede extraer el concepto de suavizado en la eliminación de ruido a costa de una borrosidad en la imagen. A medida que aumenta el tamaño de la matriz de convolución la región sobre la que actúa es mayor y en consecuencia la borrosidad de la imagen se ve seriamente afectada.

5.4.8 Filtro de conservación

El filtro de conservación [5] es una técnica de reducción de ruido. El algoritmo sacrifica la eliminación de ruido para preservar los detalles de frecuencia espacial en una imagen. Es especialmente diseñado para eliminar las gotas en la imagen, por ejemplo, píxeles aislados debido a una alta o baja intensidad y es, además, menos efectivo eliminando ruido adicional de una imagen.

Al igual que la mayoría de filtros de eliminación de ruido, el filtro de conservación opera sobre la suposición de que el ruido tiene una frecuencia espacial alta y, además, puede ser atenuado por una operación local que hace que la intensidad de cada píxel sea consistente con la de sus vecinos próximos. Sin embargo, considerando que el filtro de la media cumple esto mediante un promediado local de la intensidad, y el filtro de la mediana mediante una técnica de selección no lineal, el filtro de conservación simplemente asegura que la intensidad de cada píxel se redondea con el rango de la intensidad definido por sus vecinos.

Este filtro se ejecuta mediante un procedimiento que primero busca el mínimo y el máximo de los valores de intensidad dentro de la región de una ventana y redondea el píxel en cuestión. Si la intensidad del píxel central se encuentra entre la intensidad del rango de sus vecinos, se pasa a la salida sin cambiar de valor. Sin embargo, si el píxel central supera el máximo , se le asigna el valor del máximo, si es menor que el mínimo valor, se le asigna el mínimo valor.

Un ejemplo podría ser la matriz de una imagen cualquiera:

23	25	26	30	40
22	**24**	**26**	**27**	35
18	**20**	**50**	**25**	34
19	**15**	**19**	**23**	33
11	16	10	20	30

Tabla 5–17 Submatriz de una imagen

Tomando los valores en negrita como submatriz a analizar:

- Valores de los vecinos: 15, 19, 20, 23, 24, 25, 26, 27.
- Valor del píxel a analizar: 50.
- Valor mínimo de los vecinos: 15.
- Valor máximo de los vecinos: 27.

El valor a asignar al píxel es 27 ya que no está en el intervalo [15-27] de sus vecinos y como se excede por arriba se le aplica el máximo.

Si comparamos el resultado obtenido mediante el filtro de conservación en un segmento de la imagen con el resultado obtenido con el filtro de la media o la mediana, podemos ver como la conservación produce un efecto mas subarrendado que ambos. Además, es menos corrupto en los bordes que los filtros de eliminación de ruido.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  ConservationFilter(C_Matrix & matrix, IndexT sizeConv).

  Calcula la matriz de la imagen a partir del rango de valores definido por sus vecinos para cada píxel a tratar.

- ***Parámetros de entrada***

  matrix matriz de la imagen de entrada.

  SizeConv tamaño de la matriz de convolución a aplicar.

- ***Parámetros de salida***

  matriz de la imagen de salida.

El programa recibe como parámetro el valor del tamaño de la matriz de convolución. Como comentamos anteriormente, el filtro de la conservación obtiene el valor mínimo y máximo de la ventana correspondiente al píxel que se este tratando. Según sea el valor del píxel tratado, se le asignará al píxel de salida o el valor del píxel de entrada o el máximo de los valores de la región o el mínimo.

Las imágenes son muy a menudo corrompidas por ruido de muy diversas maneras, la más frecuente es un ruido adicional o ruido impulsivo. Los filtros lineales como el filtro de la media, son la primera herramienta para un degradado de la imagen para la eliminación de ruido. El

resultado del filtro de la conservación produce una imagen que todavía contiene algo de ruido en zonas donde los píxeles vecinos presentan un pico en la intensidad.

El filtro de conservación trabaja bien para niveles bajos de ruido del estilo de gotas sobre la imagen. Sin embargo, la imagen ha sido corrompida de tal forma que más de un píxel ha sido afectado por la intensidad de sus vecinos.

Ejemplo

```
C_Image image,image2;
C_Matrix::IndexT sizeConv;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

image2 = image;
C_Trace("Conservation Filtering ...");
image.ConservationFilter(image2,sizeConv);

C_Trace ("Writting Filter Image...");
image.WriteBMP (OutFile);

image.Free();
image2.Free();
```

Resultados

En los resultados obtenidos y mostrados en las siguientes páginas, se observa como en ambas imágenes la aplicación resulta mínima. Es decir, el nombre de conservación del filtro lo define de manera precisa. Este filtro está pensado en imágenes donde aparecen ruidos o imperfecciones aisladas y permite una visualización perfecta de la imagen sin mostrar ningún tipo de modificación en las siluetas que contenga.

Los resultados obtenidos para las distintas imágenes son los siguientes:

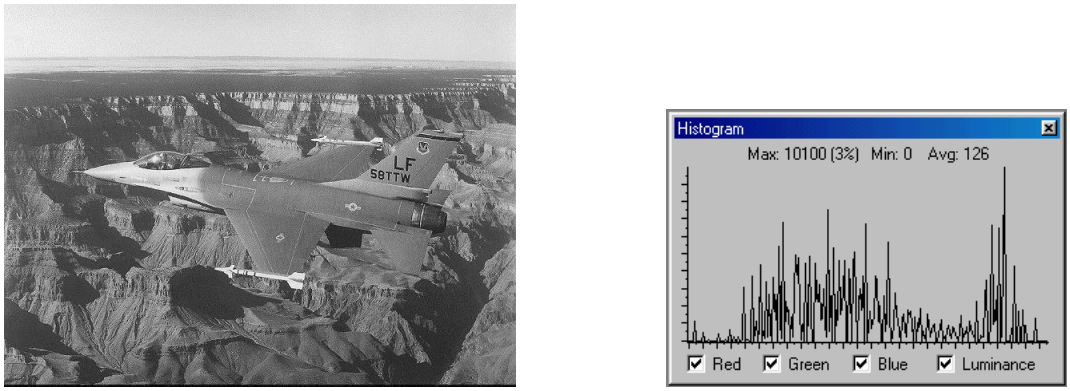


Figura 5–122 Imagen original ‘aviongr.bmp’ y su histograma

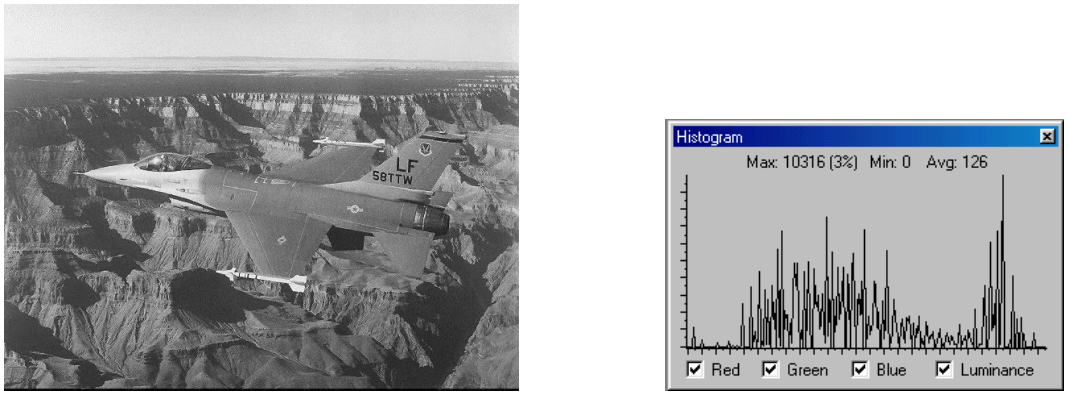


Figura 5–123 Imagen filtrada por conservación (3 × 3) ‘aviongr.bmp’ y su histograma

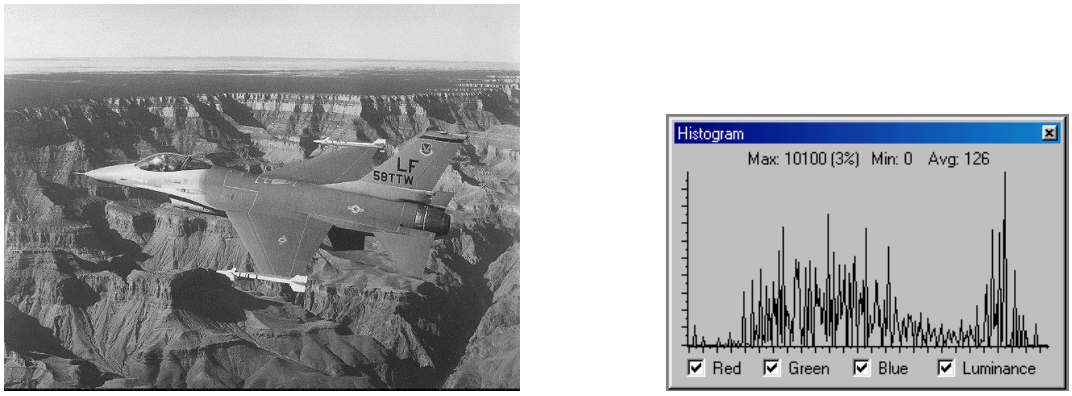


Figura 5–124 Imagen filtrada por conservación (29 × 29) ‘aviongr.bmp’ y su histograma

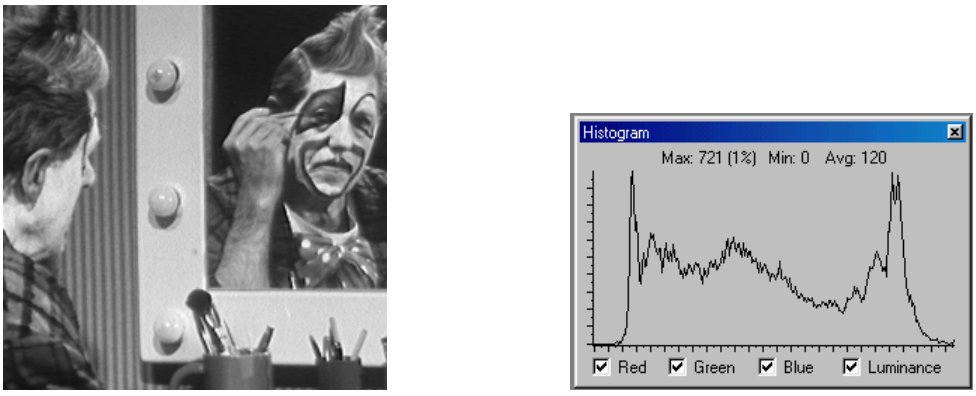


Figura 5–125 Imagen original ‘payaso.bmp’ y su histograma

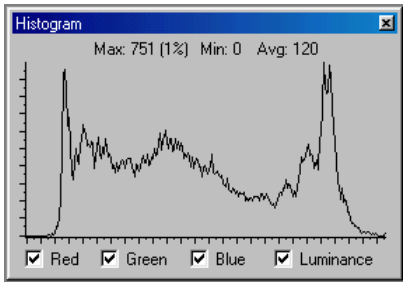


Figura 5–126 Imagen filtrada por conservación (3 × 3) 'payaso.bmp' y su histograma

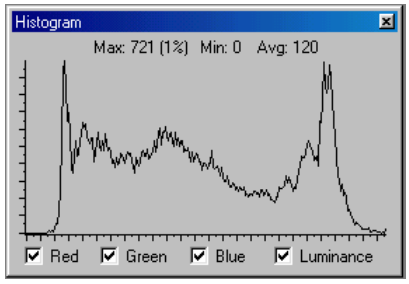


Figura 5–127 Imagen filtrada por conservación (29 × 29) 'payaso.bmp' y su histograma

5.4.9 Filtro de Crimmins

Este filtro se utiliza para la eliminación de manchas mediante un filtrado de la imagen [5] a través de un filtro de eliminación de manchas que utiliza una técnica que consiste en elevar el valor de determinados píxeles que son más oscuros que sus vecinos. De esta manera se reduce las manchas de la imagen. El algoritmo utiliza técnicas de eliminación de ruido no lineal que compara la intensidad de cada píxel en una imagen con sus vecinos, basándose en un valor relativo, incrementa o decrementa el valor del píxel tratado de tal forma que llegue a ser el más representativo del redondeo. El procedimiento de alteración del píxel utilizado por Crimmins es mas complicado que el procedimiento de ordenación utilizado por el filtro de la mediana. Envuelve una serie de pares de operaciones en las cuales el valor de la mitad del píxel en cada ventana es comparado con sus vecinos para buscar picos de intensidad.

El operador tiene una forma simple. Suponemos que a la misma vez tres píxeles consecutivos están siendo examinados como a, b y c. El algoritmo es el siguiente:

Para cada iteración se puede elegir entre un de los dos procedimientos que se muestran:

1. Ajustamos el píxel a oscuro: Para cada una de las cuatro direcciones

1. Procesamos toda la imagen: Si a ≥ b + 2 entonces b = b + 1
2. Procesamos toda la imagen: Si a > b y b ≤ c entonces b = b + 1
3. Procesamos toda la imagen: Si c > b y b ≤ c entonces b = b + 1
4. Procesamos toda la imagen: Si c ≥ b + 2 entonces b = b + 1

2. Ajustamos el píxel a claro: Para cada una de las cuatro direcciones

1. Procesamos toda la imagen: Si a ≤ b – 2 entonces b = b - 1
2. Procesamos toda la imagen: Si a < b y b ≥ c entonces b = b - 1
3. Procesamos toda la imagen: Si c < b y b ≥ a entonces b = b - 1
4. Procesamos toda la imagen: Si c ≤ b – 2 entonces b = b -1

Las cuatro direcciones a analizar en la matriz son:

Para todas las iteraciones, el efecto de filtrar de esta manera elimina los picos de intensidad. Es decir, el algoritmo filtra la imagen reduciendo la magnitud de una inconsistencia de píxel local, como también incrementa la magnitud del valor del píxel en redondeo con los valores de sus vecinos.

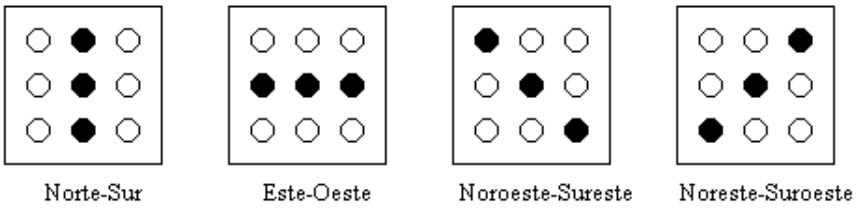


Figura 5–128 Direcciones posibles para analizar los vecinos de un píxel

Básicamente, para realizar la implementación del filtro de *Crimmins* solamente tenemos que pasar a C++ el algoritmo explicado anteriormente.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  CrimminsFilter(C_Matrix & matrix, int isDark).

  Calcula la matriz de la imagen a partir del filtrado de Crimmins y el parámetro si se desea esclarecer o oscurecer la imagen.

- ***Parámetros de entrada***

  matrix matriz de la imagen de entrada.

  isDark parámetro que indica si se desea aclarar o oscurecer la imagen.

- ***Parámetros de salida***

  matriz de la imagen de salida.

Para cada píxel se debe calcular la región de la ventana correspondiente al entorno del píxel que esta siendo tratado en cada iteración. Sobre esta matriz y según el parámetro de entrada *isDark* (1-oscurecer, 0-aclarar), se aplica la función *AdjustDark()* o *AdjustLight()*.

La interfaz de acceso al procedimiento

- ***Procedimiento***

  AdjustDark(C_Matrix ::ElementT a, C_Matrix::ElementT b, C_Matrix:: ElementT c).

  AdjustLight(C_Matrix::ElementT a, C_Matrix::ElementT b, C_Matrix:: ElementT c).

  Calcula el valor del píxel a aplicar en función de los parámetros de entrada.

- ***Parámetros de entrada***

  a b c valores correspondientes a los píxel vecinos analizados.

- ***Parámetros de salida***

  valor de salida obtenido en función del tipo de procedimiento y que se asigna al píxel tratado.

Ejemplo

```
C_Image image,image2;
C_Matrix::IndexT rowN,colN,colorsN;
int isDark;

C_Trace ("Reading Image...");
image.ReadBMP (InFile);
C_IfError (image.Fail(), "Could not read the image file", return -1);

image2 = image;
C_Trace("Crimmins Speckel Removing ...");
image.CrimminsFilter(image2,isDark);

C_Image::BMPFileInfo(InFile,rowN,colN,colorsN);
C_Trace("Strech Image");
image.Stretch(0,(colorsN-1));
```

```
C_Trace ("Writting Filter Image...");
image.WriteBMP (OutFile);

image.Free();
image2.Free();
```

Resultados

Los resultados obtenidos son los que se muestran a continuación:

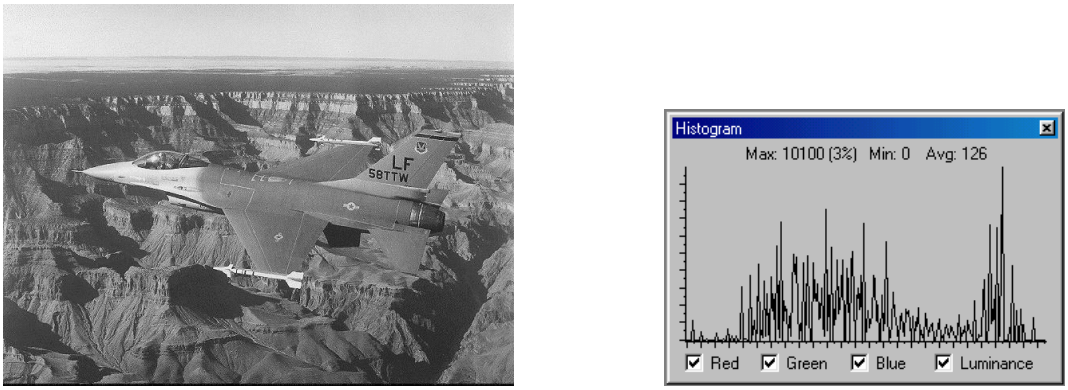


Figura 5–129 Imagen original ‘aviongr.bmp’ y su histograma

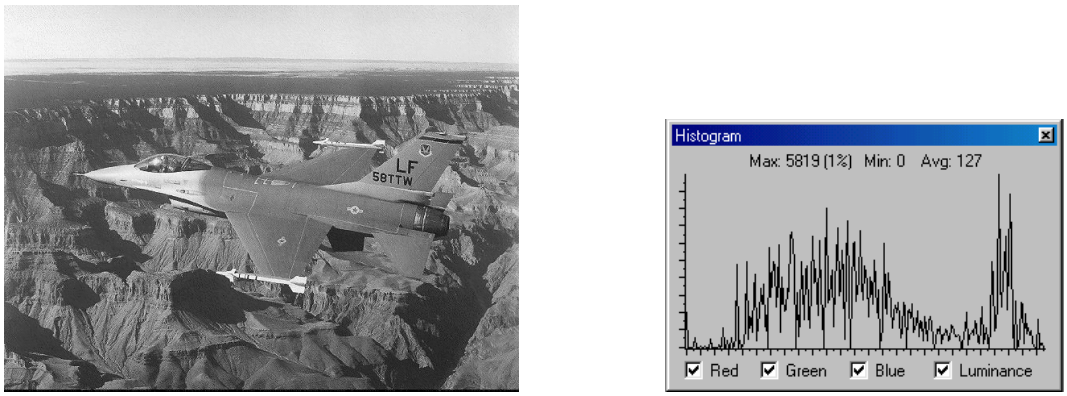


Figura 5–130 Imagen filtrada por Crimmins (Oscurecida) ‘aviongr.bmp’ y su histograma

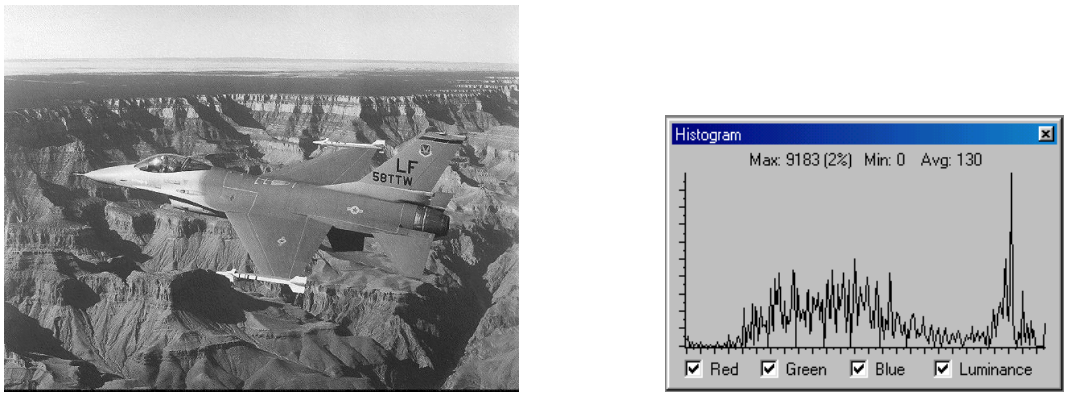


Figura 5–131 Imagen filtrada por Crimmins (Esclarecida) ‘aviongr.bmp’ y su histograma

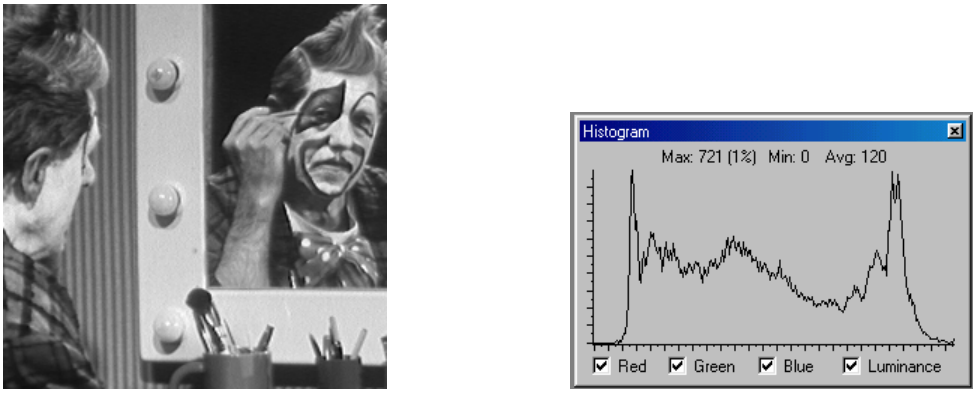


Figura 5–132 Imagen original ‘payaso.bmp’ y su histograma

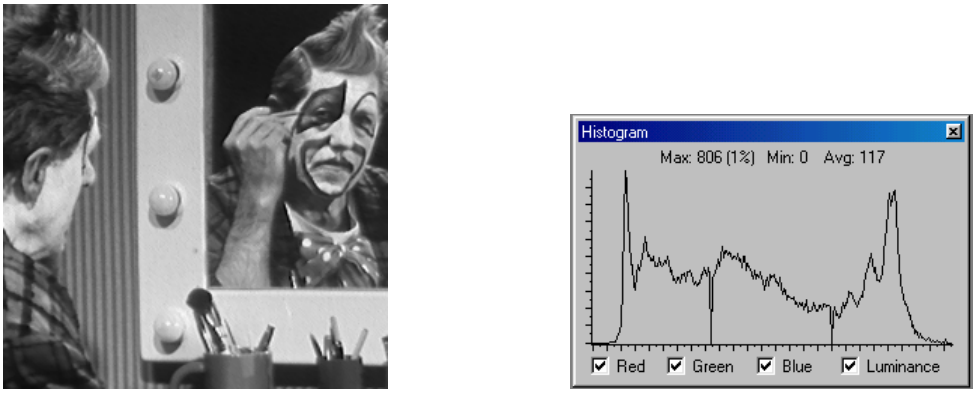


Figura 5–133 Imagen filtrada por Crimmins (Oscurecida) ‘payaso.bmp’ y su histograma

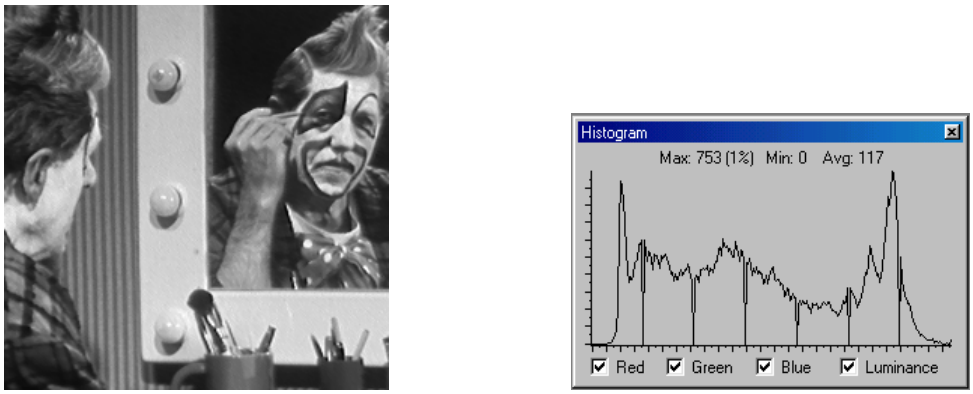


Figura 5–134 Imagen filtrada por Crimmins (Esclarecida) ‘payaso.bmp’ y su histograma

Los resultados obtenidos para el avión son un tanto confusos puede ser por el tipo de imagen en si donde hay mucho contraste entre la diversidad de objetos que la contienen. En cambio la funcionalidad del algoritmo se aprecia mejor en la imagen del payaso donde al oscurecerla la intensidad en el histograma se observa como se incrementan las regiones de color más oscuro y al esclarecer se reducen los tonos oscuros incrementándose en la misma proporción los claros.

5.4.10 Filtro diferencial

El promediado de los píxeles de una región [2] tiende a difuminar la imagen. La operación del diferencial es análoga a la integración, es de esperar que la diferenciación tenga el efecto contrario, el de aumentar la nitidez de la imagen.

El método más común de diferenciación en las aplicaciones de procesado de las imágenes es el gradiente. Para una función f(x, y), el gradiente de f en el punto de coordenadas (x, y) se define como el vector:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Ecuación 5–23 Operador gradiente

$$\overline{\nabla} f = \left[\left(\frac{\partial f}{\partial x} \right)^2 \quad \left(\frac{\partial f}{\partial y} \right)^2 \right]^{\frac{1}{2}}$$

Ecuación 5–24 Módulo del operador gradiente

es la base de las varias aproximaciones a la diferenciación de la imagen.

Para una matriz de 3 × 3, como la que se muestra a continuación:

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

la ecuación anterior puede aproximarse alrededor del punto z_5 de distintas formas. La más simple es $\nabla f \approx \left[(z_5 - z_8)^2 + (z_5 - z_6)^2 \right]^{\frac{1}{2}}$. En lugar de emplear cuadrados y raíces cuadradas, es posible obtener resultados similares empleando valores absolutos $\nabla f \approx |z_5 - z_8| + |z_5 - z_6|$. Otra posible solución a la aproximación de la ecuación anterior, consiste en efectuar diferencias cruzadas: $\nabla f \approx \left[(z_5 - z_9)^2 + (z_6 - z_8)^2 \right]^{\frac{1}{2}}$, de forma similar mediante valores absolutos sería $\nabla f \approx |z_5 - z_9| + |z_6 - z_8|$.

Otra aproximación a la ecuación inicial, con la salvedad que ahora se usa todo el entorno 3 × 3, es $\nabla f \approx |(z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)| + |(z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)|$.

La diferencia entre la tercera y la primera fila da una aproximación de la derivada en la dirección x, y la diferencia entre la tercera y la primera columna lo hace sobre la derivada en la dirección y. A continuación se muestran una serie de posibles patrones de matrices definidos por distintos autores, a los que se podría aplicar la función anterior.

Resultados

Existen distintos tipos de operadores para el cálculo del gradiente entre los que cabe destacar: los operadores de Sobel y de Prewitt. A continuación se especifican cada uno de los operadores y los resultados obtenidos con ambos.

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Tabla 5–18 Operadores de Sobel

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Tabla 5–19 Operadores de Prewitt

Los resultados por la aplicación de los distintos operadores se muestra a continuación.

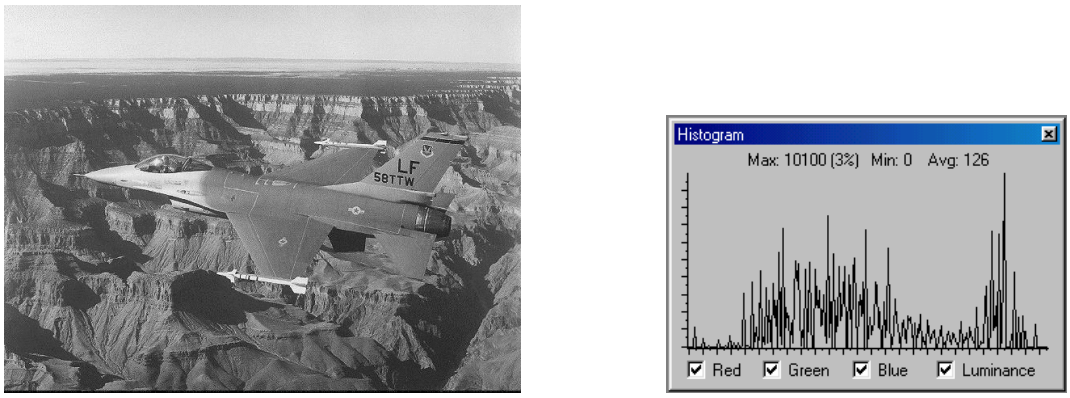


Figura 5–135 Imagen original ‘aviongr.bmp’ y su histograma

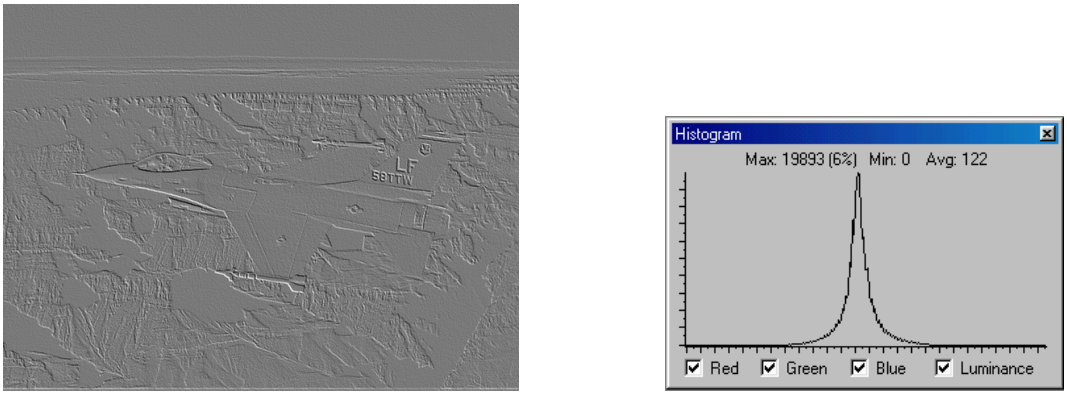


Figura 5–136 Imagen filtrada por los operadores de Sobel ‘aviongr.bmp’ y su histograma

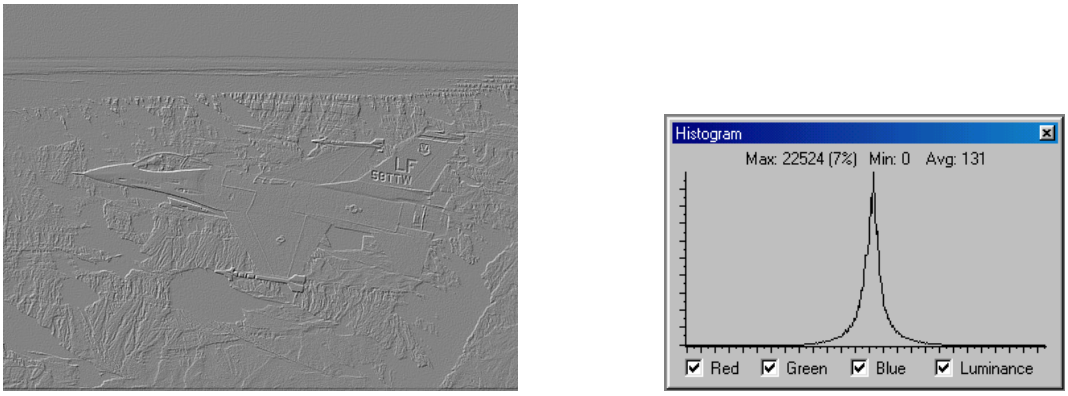


Figura 5–137 Imagen filtrada por los operadores de Prewitt ‘aviongr.bmp’ y su histograma

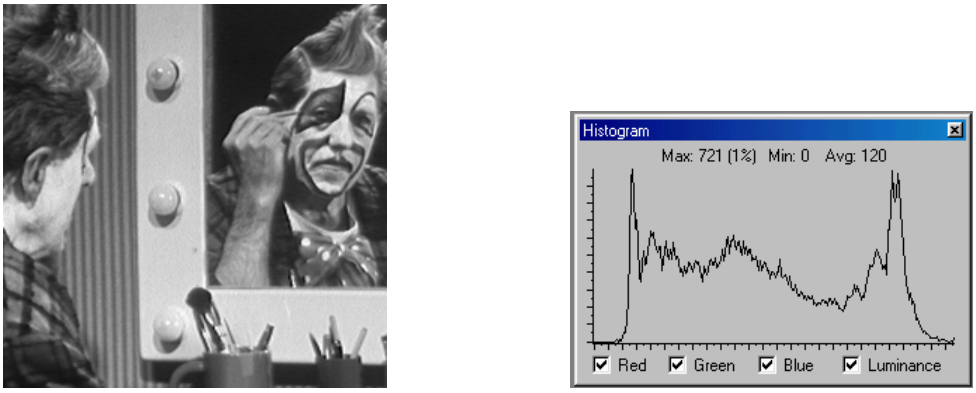


Figura 5–138 Imagen original ‘payaso.bmp’ y su histograma

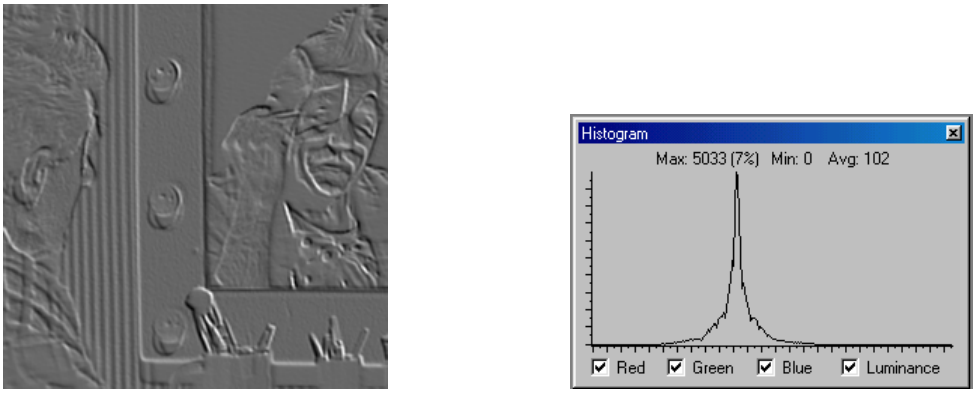
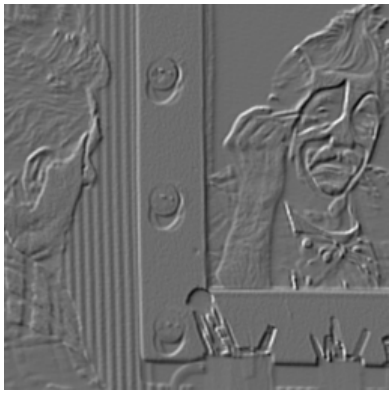


Figura 5–139 Imagen filtrada por los operadores de Sobel ‘payaso.bmp’ y su histograma



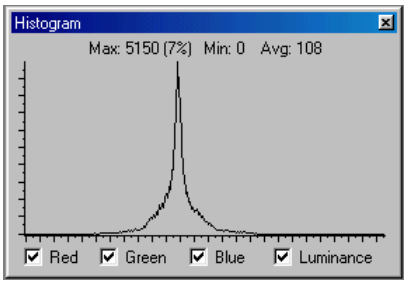


Figura 5–140 Imagen filtrada por los operadores de Prewitt 'payaso.bmp' y su histograma

Se observa como en las figuras la aplicación del operador de Sobel resalta las regiones oscuras y hunde las regiones claras, así como el operador de Prewitt produce el efecto inverso, es decir, resalta las regiones claras y hunde las regiones oscuras. Este efecto se produce principalmente por la aplicación de la función de suavizado a los niveles de gris obtenidos en el proceso. Realmente lo que se aprecia es el conjunto de contornos extraídos que posibilitan la separación de las distintas regiones en función de la diferencia de contraste entre los niveles de gris existentes.

Resultados de aplicar matrices de convolución estándar

0	1	0	-1	0
0	2	0	-2	0
0	4	0	-4	0
0	2	0	-2	0
0	1	0	-1	0

Tabla 5–20 Operador de derivación 5 × 5

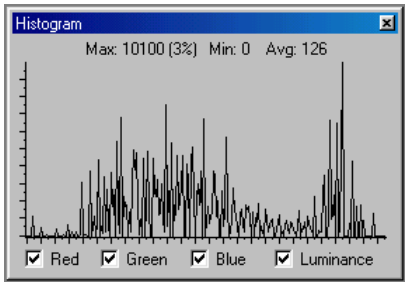


Figura 5–141 Imagen original 'aviongr.bmp' y su histograma

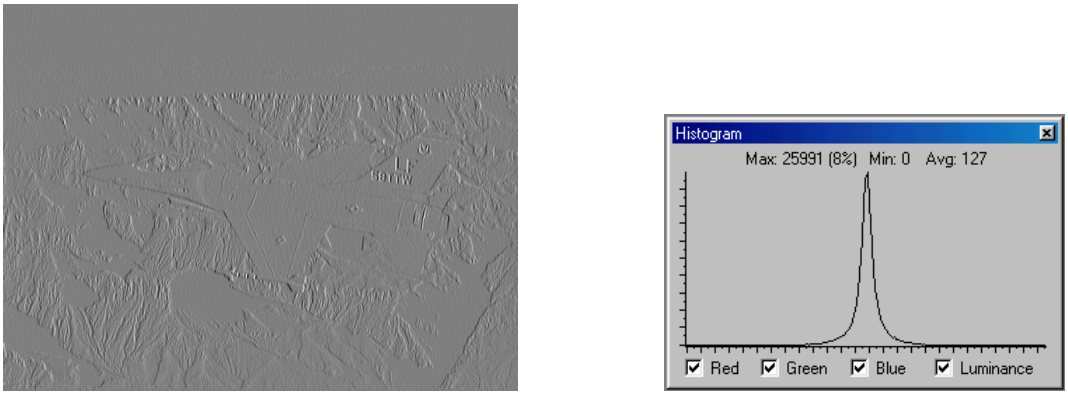


Figura 5–142 Imagen tratada por el operador derivada ‘aviongr.bmp’ y su histograma

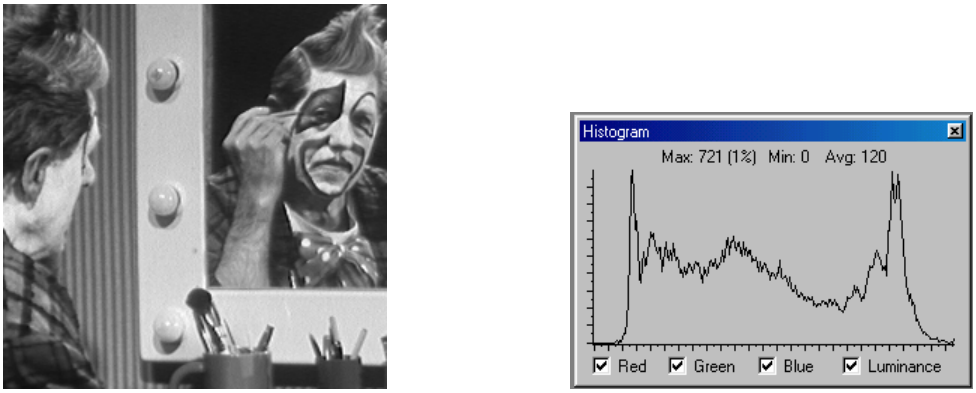


Figura 5–143 Imagen original ‘payaso.bmp’ y su histograma

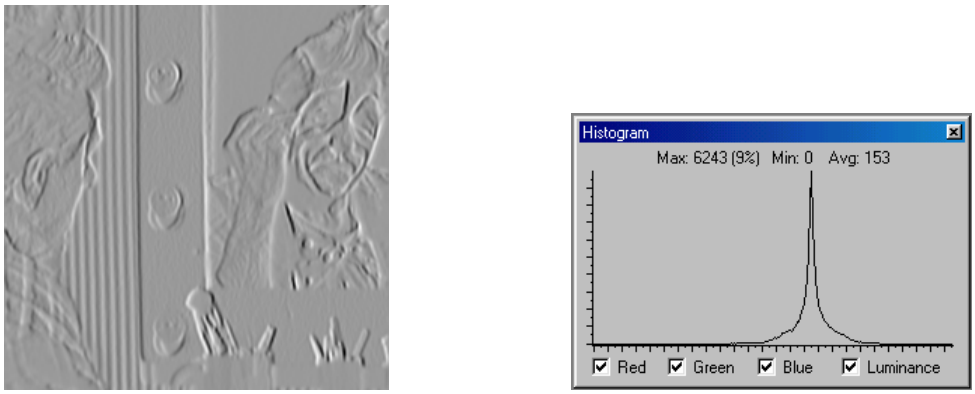


Figura 5–144 Imagen tratada por el operador derivada ‘payaso.bmp’ y su histograma

1	1	0
1	0	-1
0	-1	-1

Tabla 5–21 Operador de derivación 3 × 3

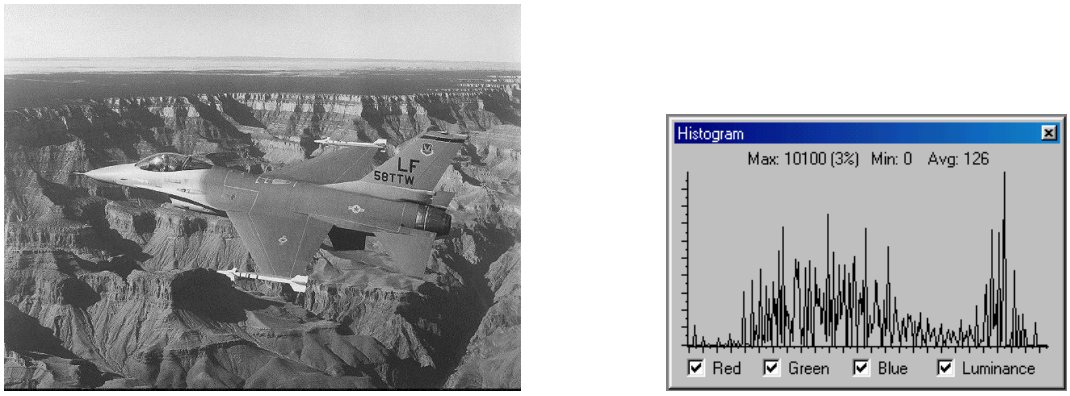


Figura 5–145 Imagen original ‘aviongr.bmp’ y su histograma

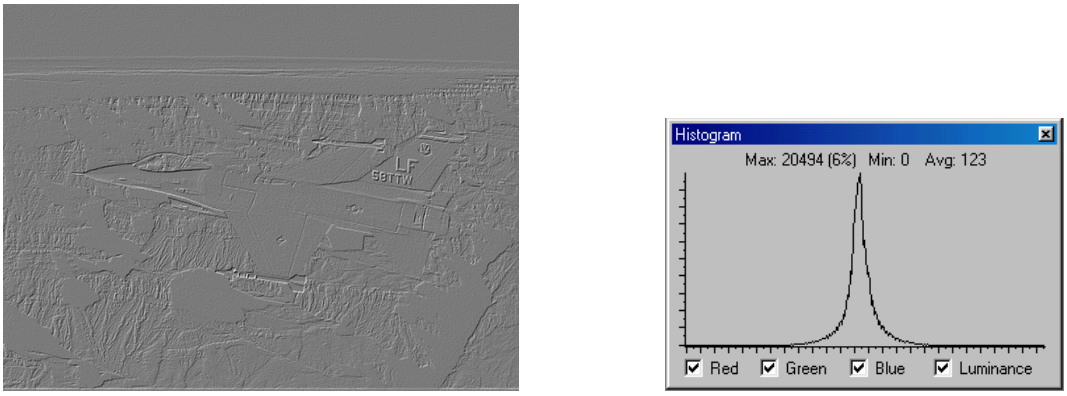


Figura 5–146 Imagen tratada por el operador derivada ‘aviongr.bmp’ y su histograma

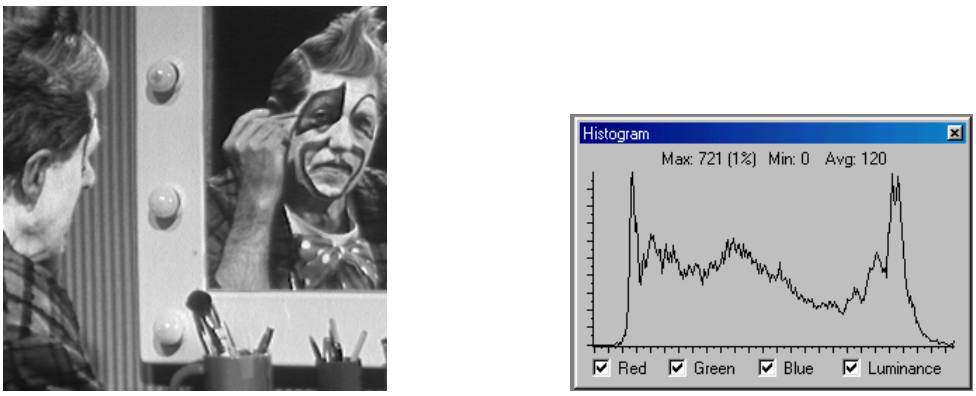


Figura 5–147 Imagen original ‘payaso.bmp’ y su histograma

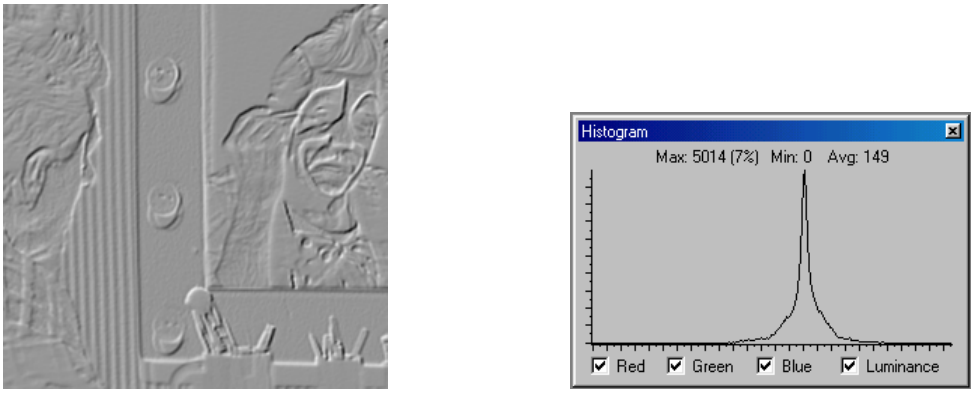


Figura 5–148 Imagen tratada por el operador derivada ‘payaso.bmp’ y su histograma

0	0	0	0	0
1	2	4	2	1
0	0	0	0	0
-1	-2	-4	-2	-1
0	0	0	0	0

Tabla 5–22 Operador de derivación 5 × 5

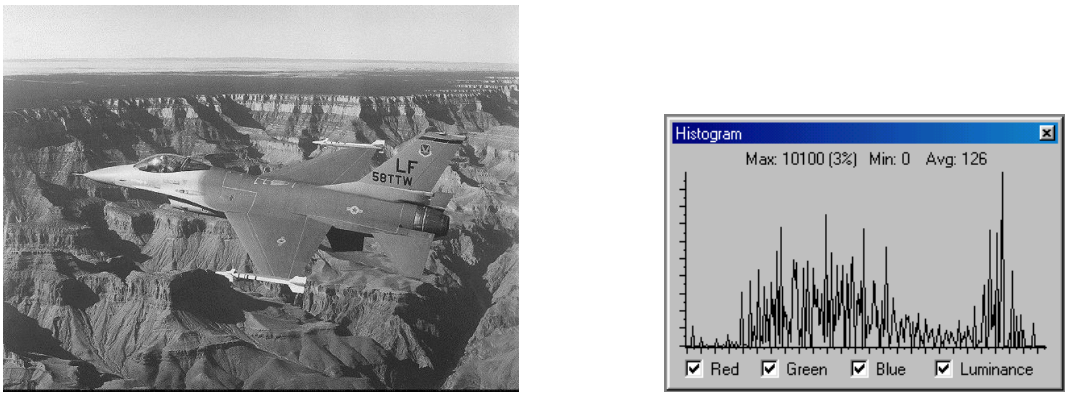


Figura 5–149 Imagen original ‘aviongr.bmp’ y su histograma

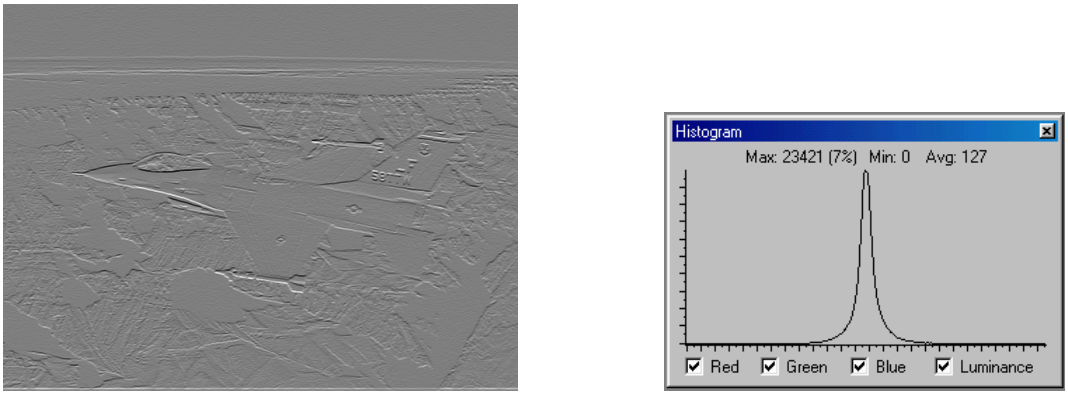


Figura 5–150 Imagen tratada por el operador derivada ‘aviongr.bmp’ y su histograma

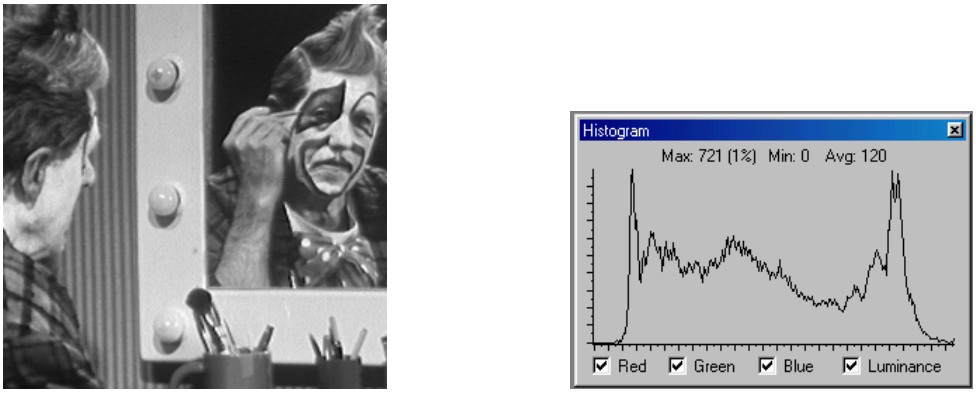
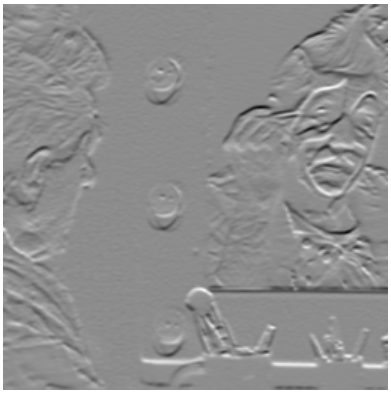


Figura 5–151 Imagen original ‘payaso.bmp’ y su histograma



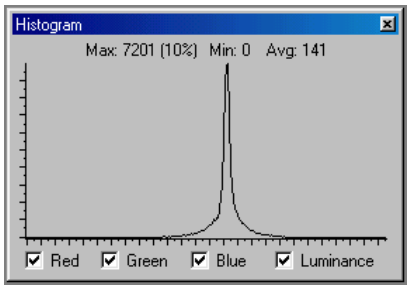


Figura 5–152 Imagen tratada por el operador derivada 'payaso.bmp' y su histograma

-0.000002	-0.000026	-0.000135	-0.000367	-0.000512	-0.000367	-0.000135	-0.000026	-0.000002
-0.000026	-0.000263	-0.001392	-0.003771	-0.005243	-0.003771	-0.001392	-0.000263	-0.000026
-0.000135	-0.001392	-0.007266	-0.017896	-0.022138	-0.017896	-0.007266	-0.001392	-0.000135
-0.000367	-0.003771	-0.017896	-0.011397	+0.041073	-0.011397	-0.017896	-0.003771	-0.000367
-0.000512	-0.005243	-0.022138	+0.041073	+0.212207	+0.042073	-0.022138	-0.005243	-0.000512
-0.000367	-0.003771	-0.017896	-0.011397	+0.041073	-0.011397	-0.017896	-0.003771	-0.000367
-0.000135	-0.001392	-0.007266	-0.017896	-0.022138	-0.017896	-0.007266	-0.001392	-0.000135
-0.000026	-0.000263	-0.001392	-0.003771	-0.005243	-0.003771	-0.001392	-0.000263	-0.000026
-0.000002	-0.000026	-0.000135	-0.000367	-0.000512	-0.000367	-0.000135	-0.000026	-0.000002

Tabla 5–23 Operador diferencial de la gaussiana 9 × 9

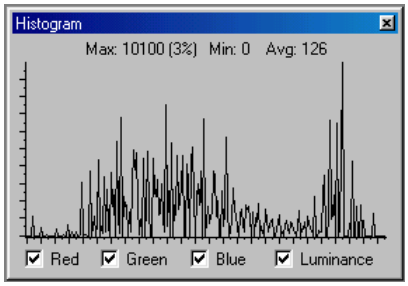
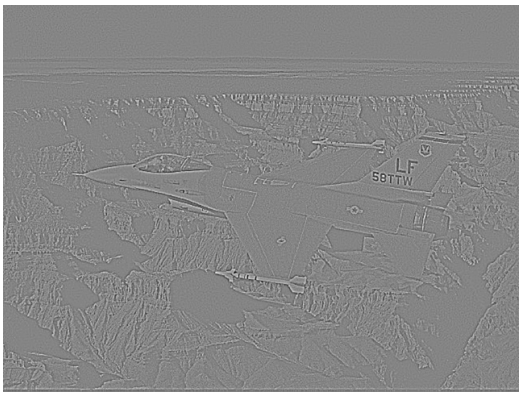


Figura 5–153 Imagen original 'aviongr.bmp' y su histograma



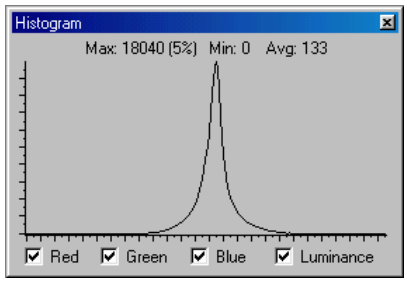


Figura 5–154 Imagen tratada por el operador diferencial de la Gaussiana 'aviongr.bmp' y su histograma.

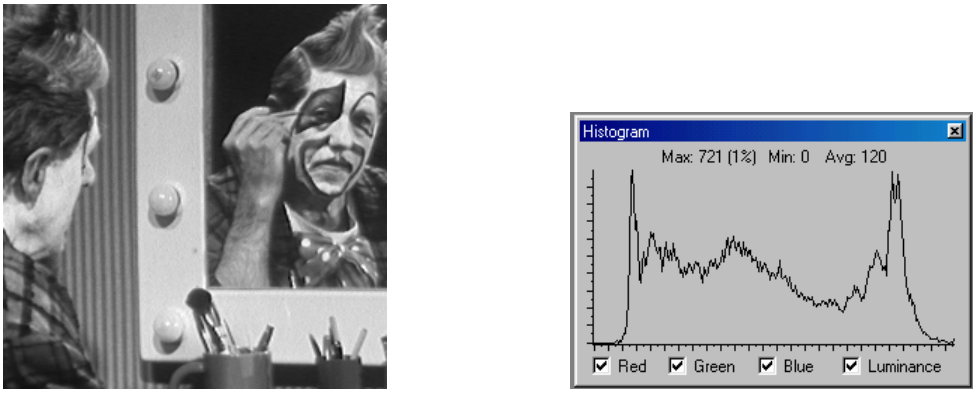


Figura 5–155 Imagen original ‘payaso.bmp’ y su histograma

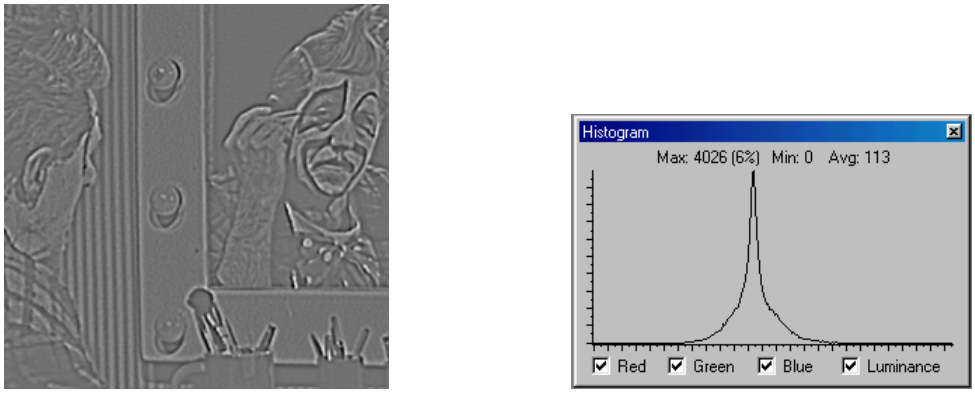


Figura 5–156 Imagen tratada por el operador diferencial de la Gaussiana ‘payaso.bmp’ y su histograma

0	-1	-2	-1	0
-1	-4	-1	-4	-1
-2	-1	36	-1	-2
-1	-4	-1	-4	-1
0	-1	-2	-1	0

Tabla 5–24 Operador DoG (Diferencial de la Gaussiana) 5 × 5

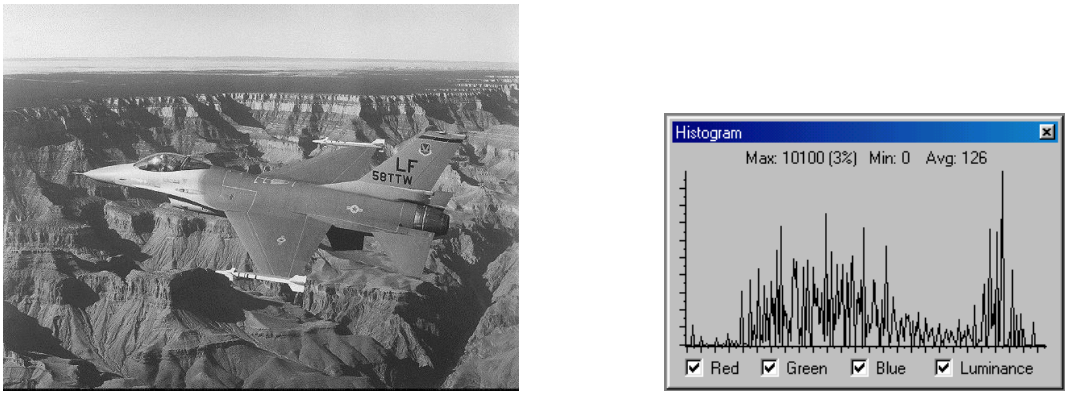


Figura 5–157 Imagen original ‘aviongr.bmp’ y su histograma

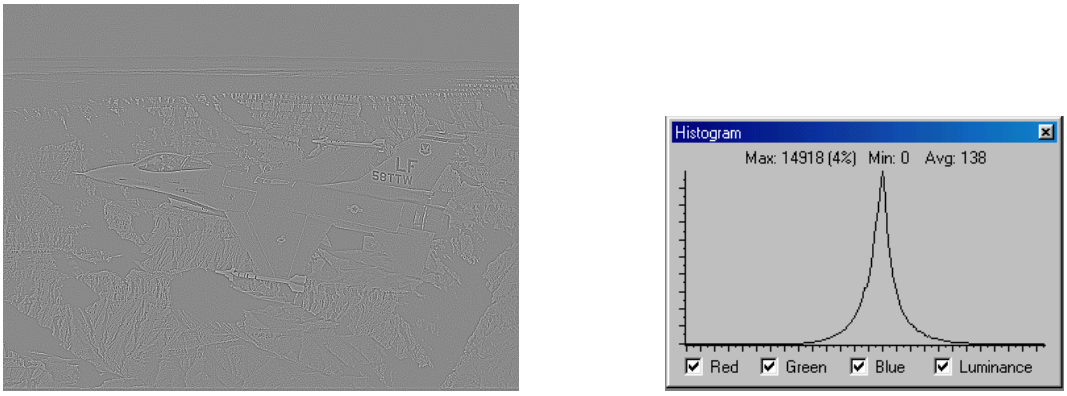


Figura 5–158 Imagen tratada por el operador DoG 5 × 5 ‘aviongr.bmp’ y su histograma

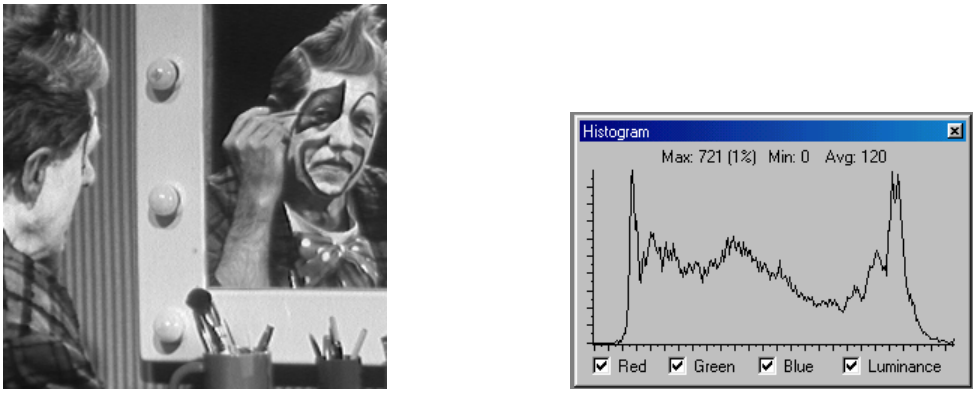


Figura 5–159 Imagen original ‘payaso.bmp’ y su histograma

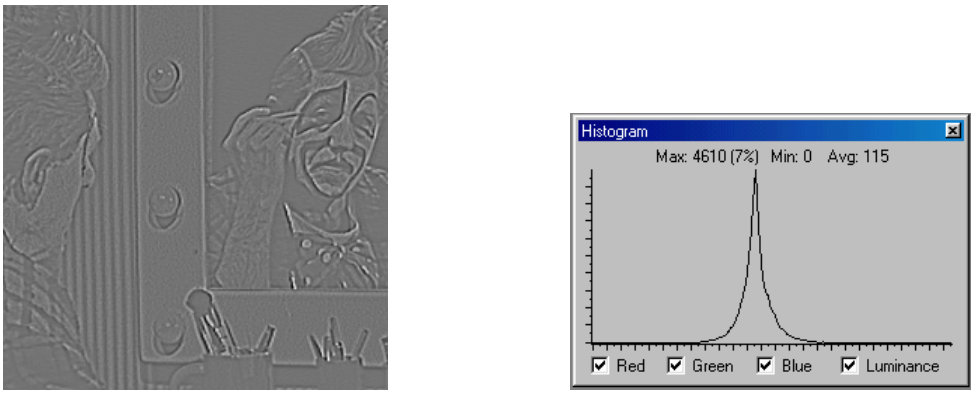


Figura 5–160 Imagen tratada por el operador DoG 5 × 5 ‘payaso.bmp’ y su histograma

0	0	0	0	-1	0	0	0	0
0	0	-1	-2	-3	-2	-1	0	0
0	-1	-4	-7	-8	-7	-4	-1	0
0	-2	-7	-2	18	-2	-7	-2	0
-1	-3	-8	18	80	18	-8	-3	-1
0	-2	-7	-2	18	-2	-7	-2	0
0	-1	-4	-7	-8	-7	-4	-1	0
0	0	-1	-2	-3	-2	-1	0	0
0	0	0	0	-1	0	0	0	0

Tabla 5–25 Operador DoG 9 × 9

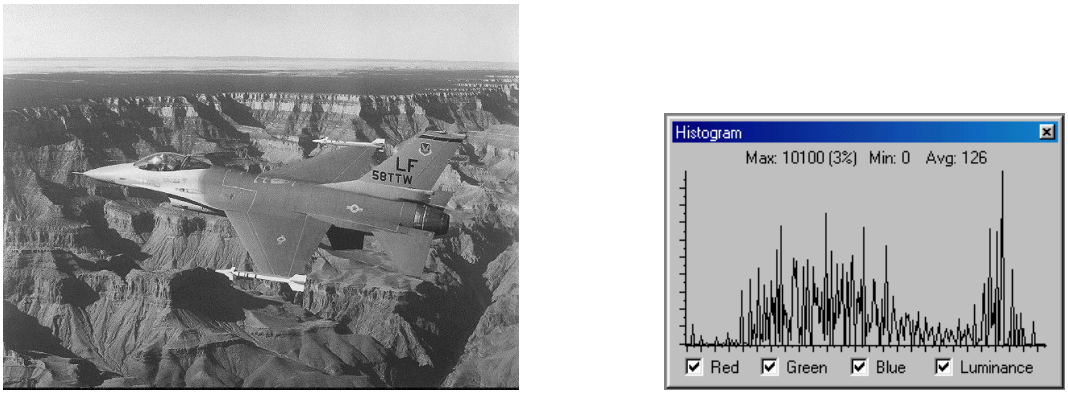


Figura 5–161 Imagen original ‘aviongr.bmp’ y su histograma

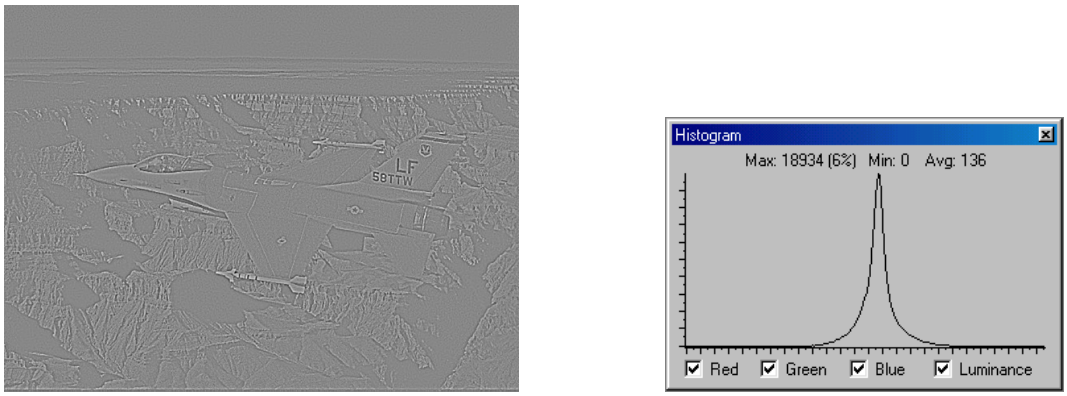


Figura 5–162 Imagen tratada por el operador DoG 9 × 9 ‘aviongr.bmp’ y su histograma

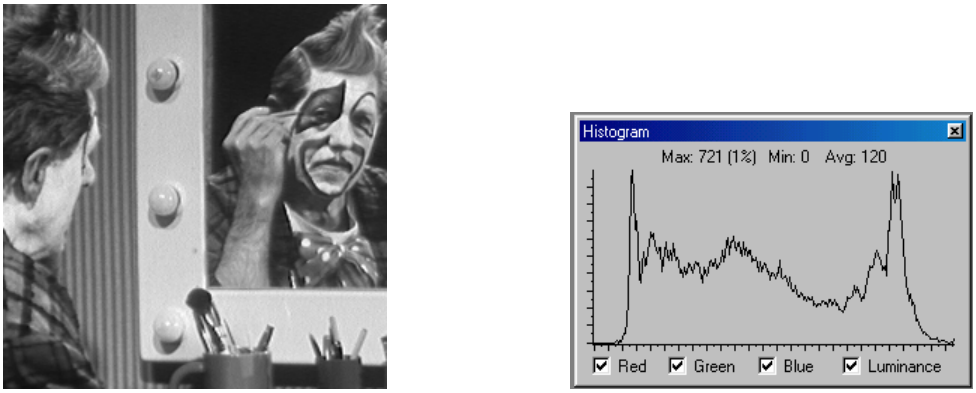


Figura 5–163 Imagen original ‘payaso.bmp’ y su histograma

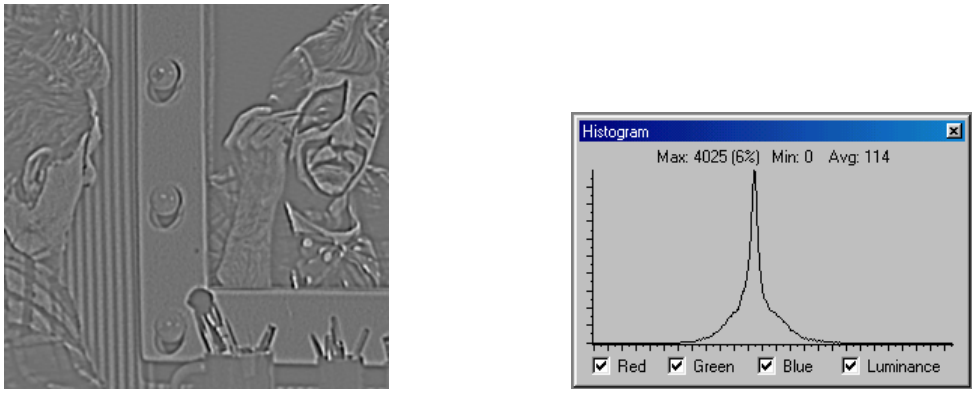


Figura 5–164 Imagen tratada por el operador DoG 9 × 9 ‘payaso.bmp’ y su histograma

0	0	0	-1	-1	-1	-1	-1	0	0	0
0	-1	-1	-2	-3	-4	-3	-2	-1	-1	0
0	-1	-3	-5	-7	-8	-7	-5	-3	-1	0
-1	-2	-5	-8	-6	-2	-6	-8	-5	-2	-1
-1	-3	-7	-6	18	41	18	-6	-7	-3	-1
-1	-4	-8	-2	41	80	41	-2	-8	-4	-1
-1	-3	-7	-6	18	41	18	-6	-7	-3	-1
-1	-2	-5	-8	-6	-2	-6	-8	-5	-2	-1
0	-1	-3	-5	-7	-8	-7	-5	-3	-1	0
0	-1	-1	-2	-3	-4	-3	-2	-1	-1	0
0	0	0	-1	-1	-1	-1	-1	0	0	0

Tabla 5–26 Operador DoG 11 × 11

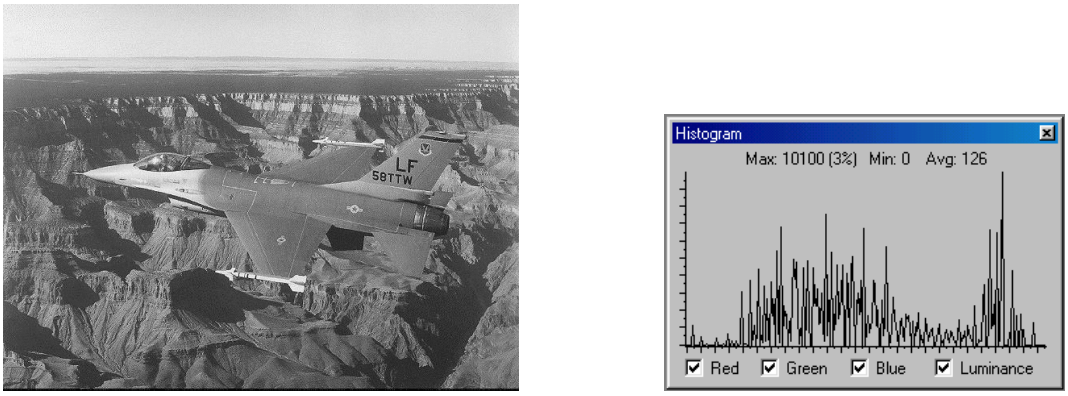


Figura 5–165 Imagen original ‘aviongr.bmp’ y su histograma

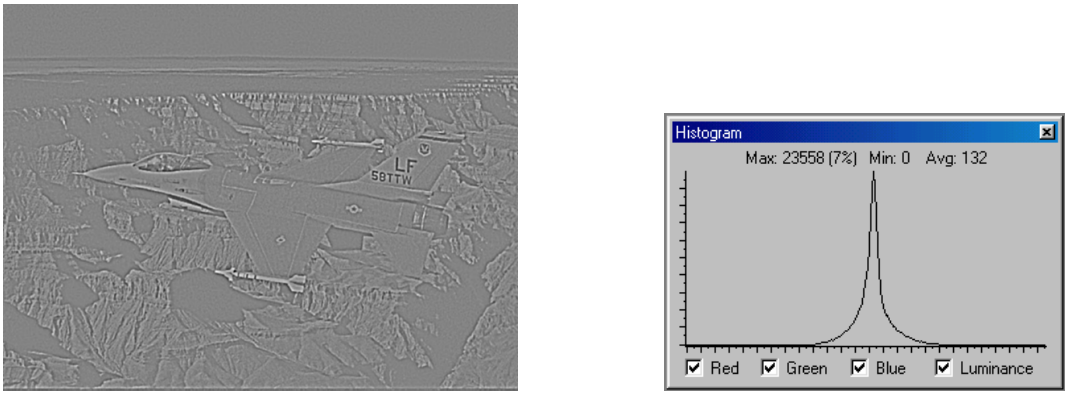


Figura 5–166 Imagen tratada por el operador DoG 11 × 11 ‘aviongr.bmp’ y su histograma

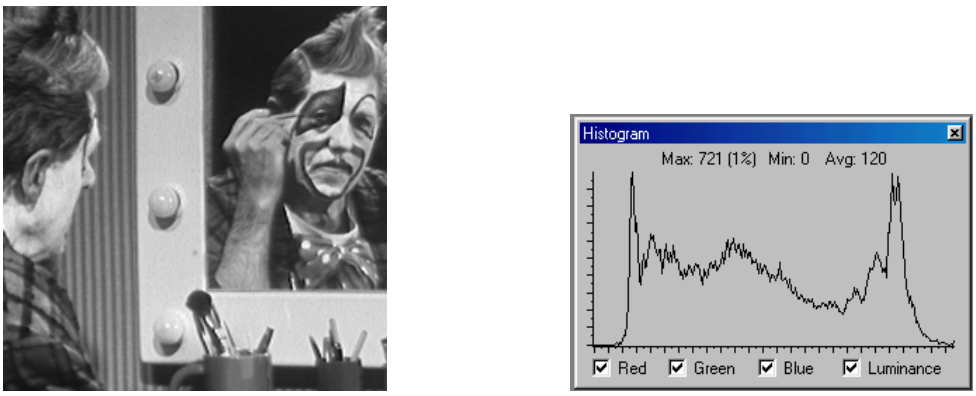


Figura 5–167 Imagen original ‘payaso.bmp’ y su histograma

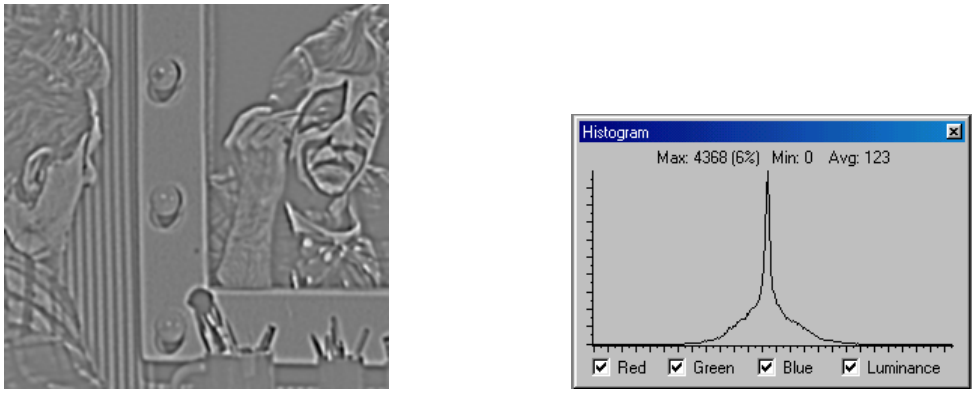


Figura 5–168 Imagen tratada por el operador DoG 11 × 11 ‘payaso.bmp’ y su histograma

0	0	0	0	-1	-1	-1	-1	-1	-1	-1	0	0	0	0
0	0	-1	-1	-2	-2	-3	-3	-3	-2	-2	-1	-1	0	0
0	-1	-1	-2	-3	-4	-5	-5	-5	-4	-3	-2	-1	-1	0
0	-1	-2	-3	-5	-7	-7	-8	-7	-7	-5	-3	-2	-1	0
-1	-2	-3	-5	-7	-7	-5	-4	-5	-7	-7	-5	-3	-2	-1
-1	-2	-4	-7	-7	-2	10	17	10	-2	-7	-7	-4	-2	-1
-1	-3	-5	-7	-5	10	38	54	38	10	-5	-7	-5	-3	-1
-1	-3	-5	-8	-4	17	54	76	54	17	-4	-8	-5	-3	-1
-1	-3	-5	-7	-5	10	38	54	38	10	-5	-7	-5	-3	-1
-1	-2	-4	-7	-7	-2	10	17	10	-2	-7	-7	-4	-2	-1
-1	-2	-3	-5	-7	-7	-5	-4	-5	-7	-7	-5	-3	-2	-1
0	-1	-2	-3	-5	-7	-7	-8	-7	-7	-5	-3	-2	-1	0
0	-1	-1	-2	-3	-4	-5	-5	-5	-4	-3	-2	-1	-1	0
0	0	-1	-1	-2	-2	-3	-3	-3	-2	-2	-1	-1	0	0
0	0	0	0	-1	-1	-1	-1	-1	-1	-1	0	0	0	0

Tabla 5–27 Operador DoG 15 × 15

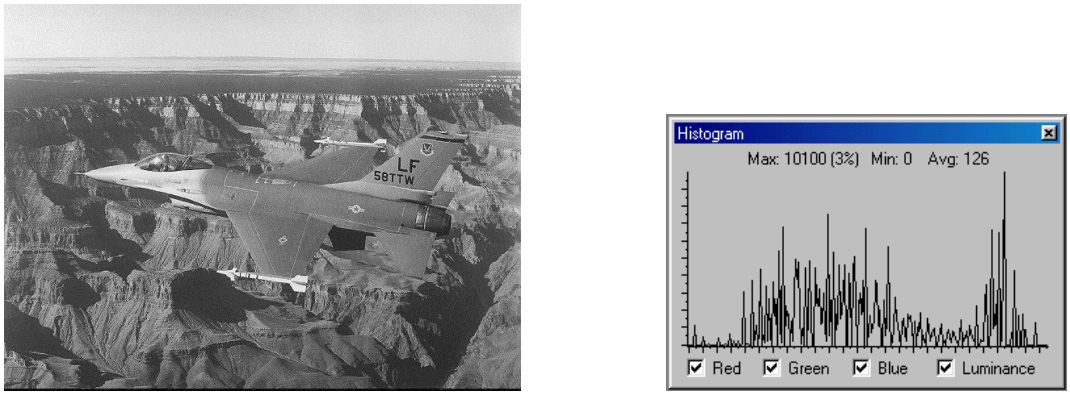


Figura 5–169 Imagen original ‘aviongr.bmp’ y su histograma

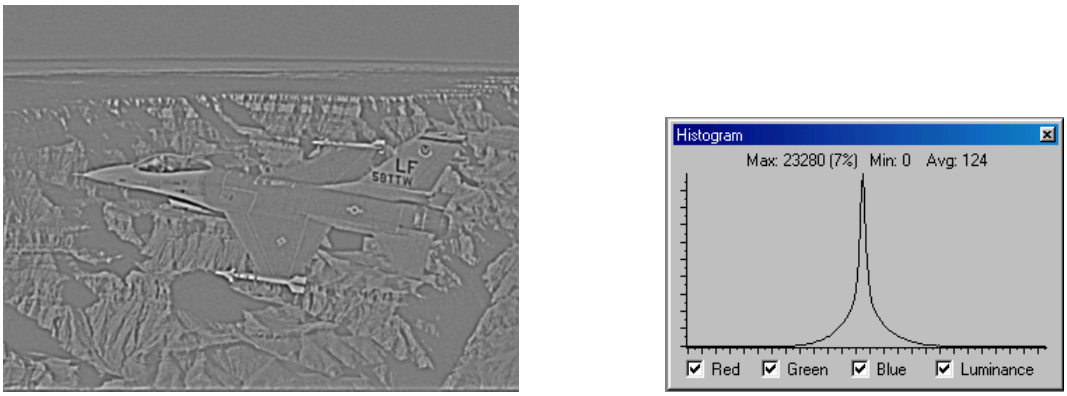


Figura 5–170 Imagen tratada por el operador DoG 15 × 15 ‘aviongr.bmp’ y su histograma

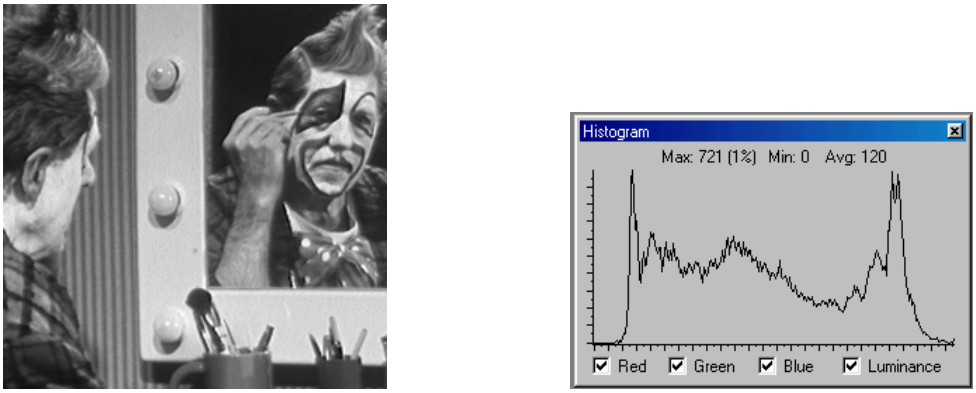


Figura 5–171 Imagen original ‘payaso.bmp’ y su histograma

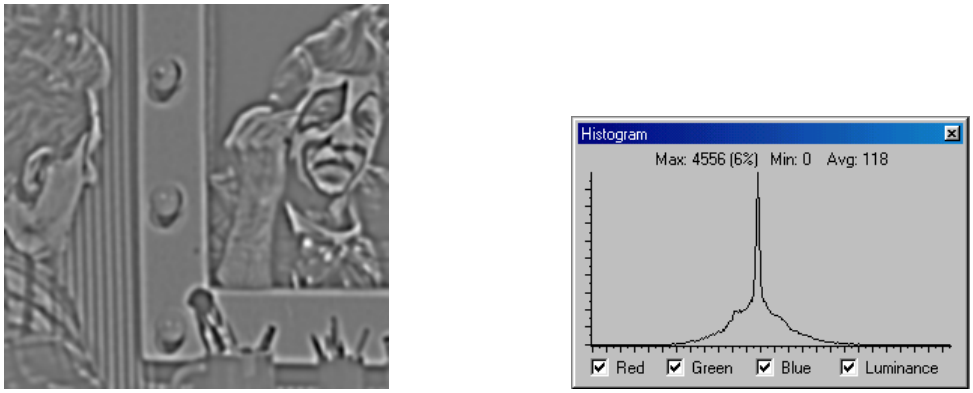


Figura 5–172 Imagen tratada por el operador DoG 15 × 15 ‘payaso.bmp’ y su histograma

Los distintos operadores diferenciales que se muestras en los ejemplos se basan en un factor muy significativa. Su principal cometido es la detección de los cambios de contraste en la imagen lo que posibilita la visualización del contorno de la misma. Es decir, en todos los ejemplos descritos se observa como el contorno de las distintas regiones que forman la imagen se encuentran más o menos bien diferenciado. Unos operadores dan mayor borrosidad a este contorno otros lo aproximan de manera más correcta . Por ejemplo, para el caso de este último operador se observa que el contorno de las distintas regiones está bien separado dando uniformidad al relleno de las mismas ya que el objetivo en sí es la detección del contorno.

6 Bibliografía

[1] DERYN GRAHAM and ANTHONY BARRETT.
Knowledge-Based Image Processing System.
Springer. (1997)

[2] GONZALEZ, R.
Digital Image Processing, Addison Wesley.
(1992).

[3] PARKER J. R.
Algorithms for image processing and computer vision.
Wiley Computer Publishing.
(1996).

[4] SONKA, M.
Image Processing, Analysis and Machine Vision, Champan & Hall.
(1995).

[5] THE DEPARTMENT OF ARTIFICIAL INTELLIGENCE IN THE UNIVERSITY OF EDINBURGH
Image processing and machine vision.
http://www.dai.ed.ac.uk/HIPR2/
Hypermedia Image Processing Reference (HIPR2).

[6] ROBERT B. FISHER
The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision.
http://www.dai.ed.ac.uk/CVonline/
Division of Informatics. University of Edinburgh.